pyglotaran Documentation

Release v0.4.0

Joern Weissenborn, Joris Snellenburg, Ivo van Stokkum

2021-06-25

CONTENTS:

CHAPTER

ONE

INTRODUCTION

Pyglotaran is a python library for global analysis of time-resolved spectroscopy data. It is designed to provide a state of the art modeling toolbox to researchers, in a user-friendly manner.

Its features are:

- user-friendly modeling with a custom YAML (`*.yml`) based modeling language
- parameter optimization using variable projection and non-negative least-squares algorithms
- easy to extend modeling framework
- battle-hardened model and algorithms for fluorescence dynamics
- build upon and fully integrated in the standard Python science stack (NumPy, SciPy, Jupyter)

1.1 A Note To Glotaran Users

Although closely related and developed in the same lab, pyglotaran is not a replacement for Glotaran - A GUI For TIMP. Pyglotaran only aims to provide the modeling and optimization framework and algorithms. It is of course possible to develop a new GUI which leverages the power of pyglotaran (contributions welcome).

The current 'user-interface' for pyglotaran is Jupyter Notebook. It is designed to seamlessly integrate in this environment and be compatible with all major visualization and data analysis tools in the scientific python environment.

If you are a non-technical user, you should give these tools a try, there are numerous tutorials how to use them. You don't need to really learn to program. If you can use e.g. Matlab or Mathematica, you can use Jupyter and Python.

CHAPTER

TWO

INSTALLATION

2.1 Prerequisites

- Python 3.6 or later

2.1.1 Windows

The easiest way of getting Python (and some basic tools to work with it) in Windows is to use Anaconda, which provides python.

You will need a terminal for the installation. One is provided by *Anaconda* and is called *Anaconda Console*. You can find it in the start menu.

Note: If you use a Windows Shell like cmd.exe or PowerShell, you might have to prefix '$PATH_TO_ANACONDA/' to all commands (e.g. *C:/Anaconda/pip.exe* instead of *pip*)

2.2 Stable release

Warning: pyglotaran is early development, so for the moment stable releases are sparse and outdated. We try to keep the master code stable, so please install from source for now.

This is the preferred method to install pyglotaran, as it will always install the most recent stable release.

To install pyglotaran, run this command in your terminal:

```
$ pip install pyglotaran
```

If you don't have pip installed, this Python installation guide can guide you through the process.

If you want to install it via conda, you can run the following command:

```
$ conda install -c conda-forge pyglotaran
```

2.3 From sources

First you have to install or update some dependencies.

Within a terminal:

```
$ pip install -U numpy scipy Cython
```

Alternatively, for Anaconda users:

```
$ conda install numpy scipy Cython
```

Afterwards you can simply use pip to install it directly from Github.

```
$ pip install git+https://github.com/glotaran/pyglotaran.git
```

For updating pyglotaran, just re-run the command above.

If you prefer to manually download the source files, you can find them on Github. Alternatively you can clone them with git (preferred):

```
$ git clone https://github.com/glotaran/pyglotaran.git
```

Within a terminal, navigate to directory where you have unpacked or cloned the code and enter

```
$ pip install -e .
```

For updating, simply download and unpack the newest version (or run `$ git pull` in pyglotaran directory if you used git) and and re-run the command above.

The following section was generated from `docs/source/notebooks/quickstart/quickstart.ipynb` ....

CHAPTER

THREE

QUICKSTART/CHEAT-SHEET

Since this documentation is written in a jupyter-notebook we will import a little ipython helper function to display file with syntax highlighting.

[1]:
```
from glotaran.utils.ipython import display_file
```

To start using `pyglotaran` in your project, you have to import it first. In addition we need to import some extra components for later use.

[2]:
```
from glotaran.analysis.optimize import optimize
from glotaran.io import load_model
from glotaran.io import load_parameters
from glotaran.io import save_dataset
from glotaran.io.prepare_dataset import prepare_time_trace_dataset
from glotaran.project.scheme import Scheme
```

Let us get some example data to analyze:

[3]:
```
from glotaran.examples.sequential import dataset

dataset
```

[3]:
```
<xarray.Dataset>
Dimensions:   (spectral: 72, time: 2100)
Coordinates:
  * time      (time) float64 -1.0 -0.99 -0.98 -0.97 ... 19.96 19.97 19.98 19.99
  * spectral  (spectral) float64 600.0 601.4 602.8 604.2 ... 696.6 698.0 699.4
Data variables:
    data      (time, spectral) float64 0.01272 -0.003198 ... 1.718 1.542
```

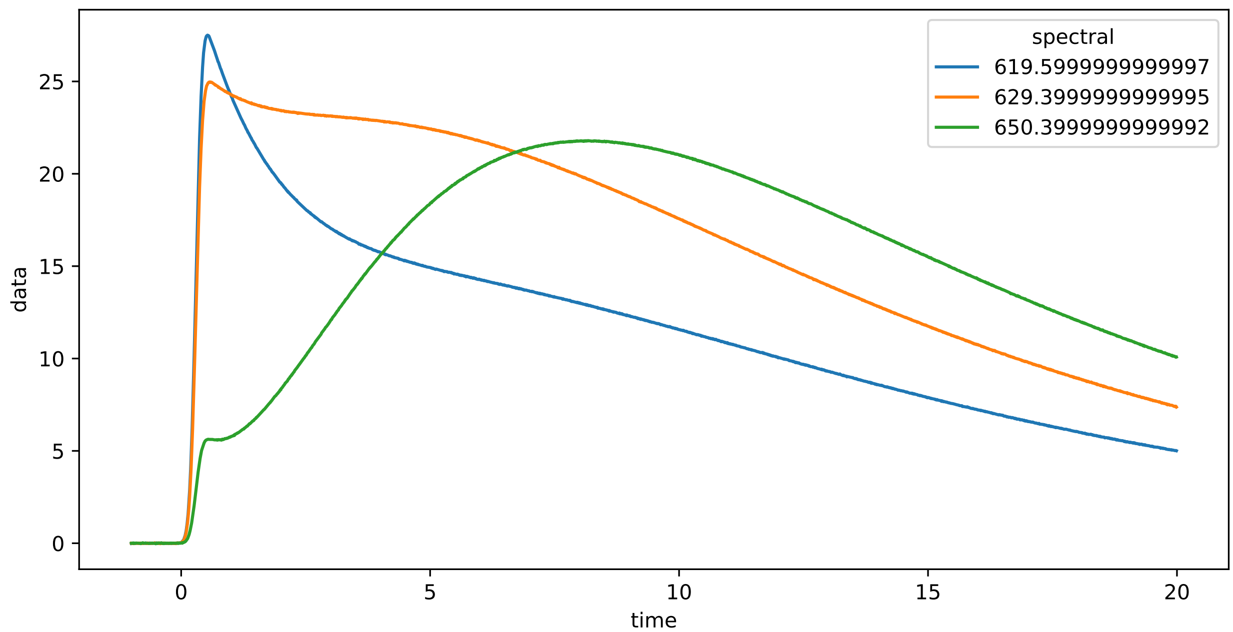
Like all data in `pyglotaran`, the dataset is a xarray.Dataset. You can find more information about the `xarray` library the xarray hompage.

The loaded dataset is a simulated sequential model.

3.1 Plotting raw data

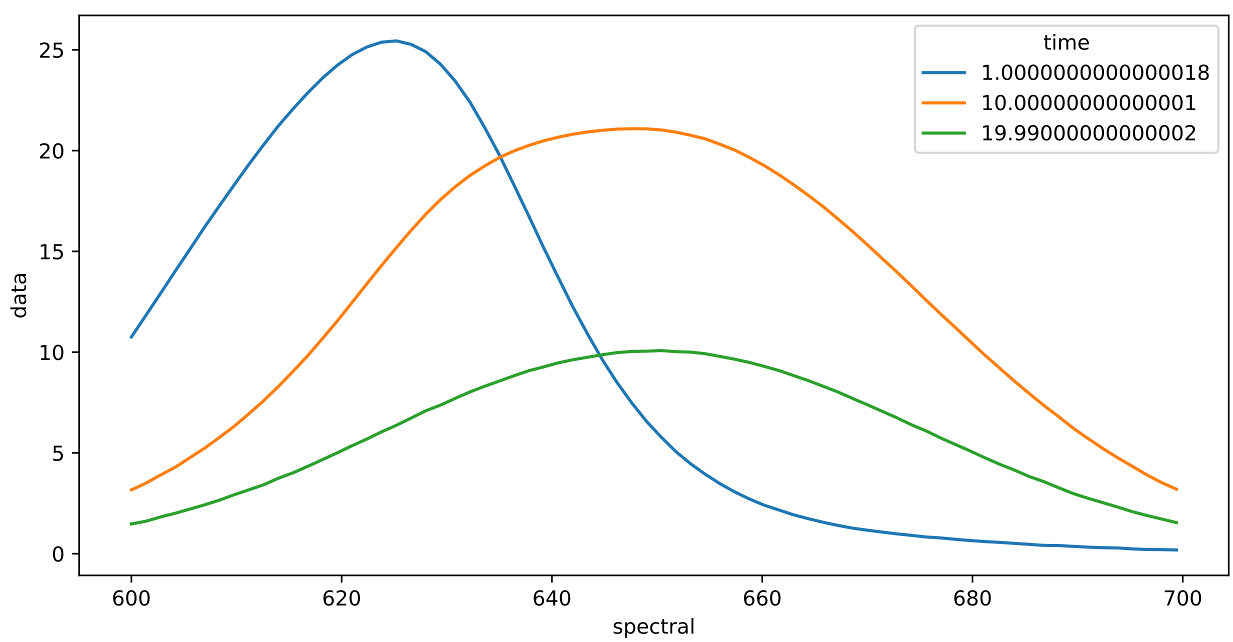
Now we lets plot some time traces.

[4]:
```
plot_data = dataset.data.sel(spectral=[620, 630, 650], method="nearest")
plot_data.plot.line(x="time", aspect=2, size=5);
```



We can also plot spectra at different times.

[5]:
```
plot_data = dataset.data.sel(time=[1, 10, 20], method="nearest")
plot_data.plot.line(x="spectral", aspect=2, size=5);
```



3.2 Preparing data

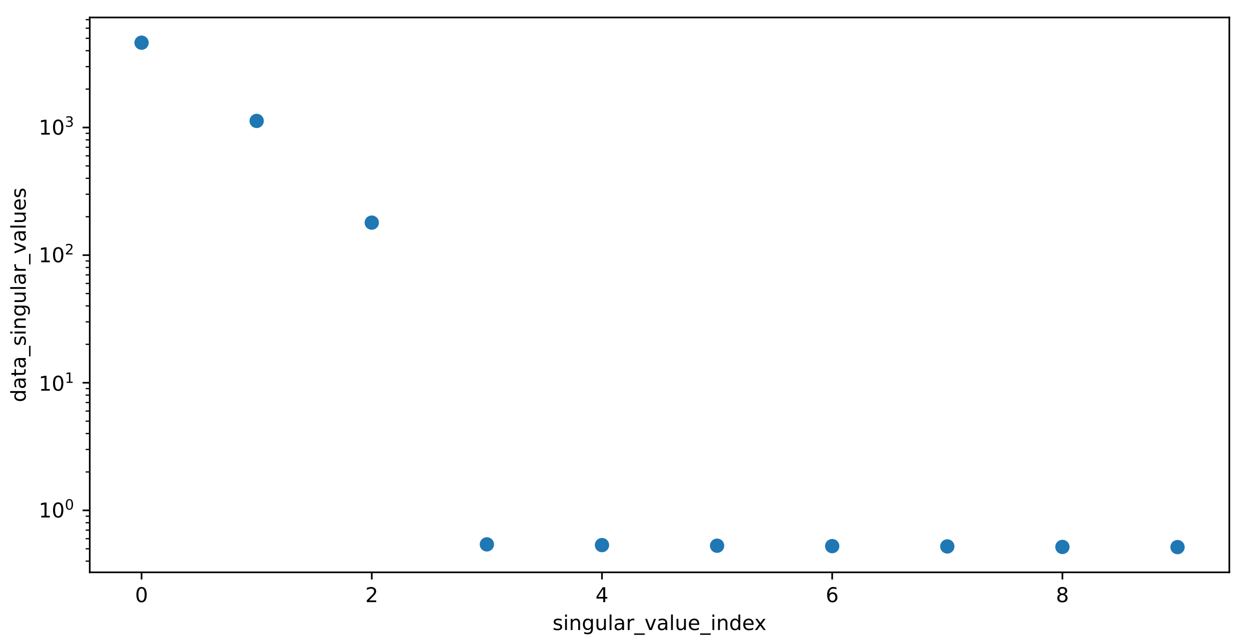
To get an idea about how to model your data, you should inspect the singular value decomposition. Pyglotaran has a function to calculate it (among other things).

[6]:
```
dataset = prepare_time_trace_dataset(dataset)
dataset
```

[6]:
```
<xarray.Dataset>
Dimensions:                      (left_singular_value_index: 72, right_singular_value_
↪index: 72, singular_value_index: 72, spectral: 72, time: 2100)
Coordinates:
  * time                         (time) float64 -1.0 -0.99 -0.98 ... 19.98 19.99
  * spectral                     (spectral) float64 600.0 601.4 ... 698.0 699.4
Dimensions without coordinates: left_singular_value_index, right_singular_value_index,␣
↪singular_value_index
Data variables:
    data                         (time, spectral) float64 0.01272 ... 1.542
    data_left_singular_vectors   (time, left_singular_value_index) float64 -2...
    data_singular_values         (singular_value_index) float64 4.62e+03 ... ...
    data_right_singular_vectors  (right_singular_value_index, spectral) float64 ...
```

First, take a look at the first 10 singular values:

[7]:
```
plot_data = dataset.data_singular_values.sel(singular_value_index=range(0, 10))
plot_data.plot(yscale="log", marker="o", linewidth=0, aspect=2, size=5);
```



3.3 Working with models

To analyze our data, we need to create a model.

Create a file called `model.yaml` in your working directory and fill it with the following:

[8]:
```
display_file("model.yaml", syntax="yaml")
```

[8]:
```yaml
type: kinetic-spectrum

initial_concentration:
  input:
    compartments: [s1, s2, s3]
    parameters: [input.1, input.0, input.0]

k_matrix:
  k1:
    matrix:
      (s2, s1): kinetic.1
      (s3, s2): kinetic.2
      (s3, s3): kinetic.3

megacomplex:
  m1:
    k_matrix: [k1]

irf:
  irf1:
    type: gaussian
    center: irf.center
    width: irf.width

dataset:
  dataset1:
    initial_concentration: input
    megacomplex: [m1]
    irf: irf1
```

Now you can load the model file.

[9]:
```
model = load_model("model.yaml")
```

You can check your model for problems with `model.validate`.

[10]:
```
model.validate()
```

[10]:
```
'Your model is valid.'
```

3.4 Working with parameters

Now define some starting parameters. Create a file called `parameters.yaml` with the following content.

[11]:
```
display_file("parameters.yaml", syntax="yaml")
```

[11]:
```yaml
input:
 - ['1', 1, {'vary': False, 'non-negative': False}]
 - ['0', 0, {'vary': False, 'non-negative': False}]

kinetic: [
    0.5,
    0.3,
    0.1,
]

irf:
 - ['center', 0.3]
 - ['width', 0.1]
```

[12]:
```
parameters = load_parameters("parameters.yaml")
```

You can `model.validate` also to check for missing parameters.

[13]:
```
model.validate(parameters=parameters)
```

[13]:
```
'Your model is valid.'
```

Since not all problems in the model can be detected automatically it is wise to visually inspect the model. For this purpose, you can just print the model.

[14]:
```
model
```

[14]:

3.4.1 Model

Type: kinetic-spectrum

Initial Concentration

- **input**:
- *Label*: input
- *Compartments*: [‘s1’, ‘s2’, ‘s3’]
- *Parameters*: [input.1, input.0, input.0]
- *Exclude From Normalize*: []

K Matrix

- **k1**:

(continues on next page)

(continued from previous page)

- *Label*: k1
- *Matrix*:
  - *('s2', 's1')*: kinetic.1
  - *('s3', 's2')*: kinetic.2
  - *('s3', 's3')*: kinetic.3

Irf

- **irf1** (gaussian):
- *Label*: irf1
- *Type*: gaussian
- *Center*: irf.center
- *Width*: irf.width
- *Normalize*: True
- *Backsweep*: False

Dataset

- **dataset1**:
- *Label*: dataset1
- *Megacomplex*: ['m1']
- *Initial Concentration*: input
- *Irf*: irf1

Megacomplex

- **m1** (None):
- *Label*: m1
- *K Matrix*: ['k1']

The same way you should inspect your parameters.

[15]:
```
parameters
```

[15]:

- **input**:

Label	*Value*	*StdErr*	*Min*	*Max*	*Vary*	*Non-Negative*	*Expr*
1	1	0	-inf	inf	False	False	None
0	0	0	-inf	inf	False	False	None

(continues on next page)

(continued from previous page)

- **irf**:

Label	*Value*	*StdErr*	*Min*	*Max*	*Vary*	*Non-Negative*	*Expr*
center	0.3	0	-inf	inf	True	False	None
width	0.1	0	-inf	inf	True	False	None

- **kinetic**:

Label	*Value*	*StdErr*	*Min*	*Max*	*Vary*	*Non-Negative*	*Expr*
1	0.5	0	-inf	inf	True	False	None
2	0.3	0	-inf	inf	True	False	None
3	0.1	0	-inf	inf	True	False	None

3.5 Optimizing data

Now we have everything together to optimize our parameters. First we import optimize.

[16]:
```
scheme = Scheme(model, parameters, {"dataset1": dataset})
result = optimize(scheme)
result
```

```
   Iteration     Total nfev        Cost      Cost reduction    Step norm     Optimality
       0              1         7.5559e+00                                    1.58e+01
       1              2         7.5557e+00      2.17e-04       8.28e-05       6.62e-02
       2              3         7.5557e+00      3.99e-11       4.37e-09       5.12e-06
Both `ftol` and `xtol` termination conditions are satisfied.
Function evaluations 3, initial cost 7.5559e+00, final cost 7.5557e+00, first-order
↪optimality 5.12e-06.
```

[16]:

Optimization Result	
Number of residual evaluation	3
Number of variables	5
Number of datapoints	151200
Degrees of freedom	151195
Chi Square	1.51e+01
Reduced Chi Square	9.99e-05
Root Mean Square Error (RMSE)	1.00e-02

3.5.1 Model

Type: kinetic-spectrum

Initial Concentration

- **input**:
- *Label*: input

(continues on next page)

(continued from previous page)

- *Compartments*: ['s1', 's2', 's3']
- *Parameters*: [input.1: **1.00000e+00** *(fixed)*, input.0: **0.00000e+00** *(fixed)*, input.0: **0.00000e+00** *(fixed)*]
- *Exclude From Normalize*: []

K Matrix

- **k1**:
- *Label*: k1
- *Matrix*:
  - *('s2', 's1')*: kinetic.1: **5.00082e-01** *(StdErr: 7e-05 ,initial: 5.00000e-01)*
  - *('s3', 's2')*: kinetic.2: **2.99990e-01** *(StdErr: 4e-05 ,initial: 3.00000e-01)*
  - *('s3', 's3')*: kinetic.3: **9.99963e-02** *(StdErr: 5e-06 ,initial: 1.00000e-01)*

Irf

- **irf1** (gaussian):
- *Label*: irf1
- *Type*: gaussian
- *Center*: irf.center: **3.00002e-01** *(StdErr: 5e-06 ,initial: 3.00000e-01)*
- *Width*: irf.width: **1.00006e-01** *(StdErr: 7e-06 ,initial: 1.00000e-01)*
- *Normalize*: True
- *Backsweep*: False

Dataset

- **dataset1**:
- *Label*: dataset1
- *Megacomplex*: ['m1']
- *Initial Concentration*: input
- *Irf*: irf1

Megacomplex

- **m1** (None):
- *Label*: m1
- *K Matrix*: ['k1']

[17]:
```
result.optimized_parameters
```

[17]:

- **input**:

Label	Value	StdErr	Min	Max	Vary	Non-Negative	Expr
1	1	0	-inf	inf	False	False	None
0	0	0	-inf	inf	False	False	None

- **irf**:

Label	Value	StdErr	Min	Max	Vary	Non-Negative	Expr
center	0.300002	5.00976e-06	-inf	inf	True	False	None
width	0.100006	6.70229e-06	-inf	inf	True	False	None

- **kinetic**:

Label	Value	StdErr	Min	Max	Vary	Non-Negative	Expr
1	0.500082	7.2547e-05	-inf	inf	True	False	None
2	0.29999	4.18992e-05	-inf	inf	True	False	None
3	0.0999963	4.77853e-06	-inf	inf	True	False	None

You can get the resulting data for your dataset with `result.get_dataset`.

[18]:
```
result_dataset = result.data["dataset1"]
result_dataset
```

[18]:
```
<xarray.Dataset>
Dimensions:                              (clp_label: 3, component: 3, from_species:␣
↪3, left_singular_value_index: 72, right_singular_value_index: 72, singular_value_index:
↪ 72, species: 3, spectral: 72, time: 2100, to_species: 3)
Coordinates:
  * time                                 (time) float64 -1.0 ... 19.99
  * spectral                             (spectral) float64 600.0 ... 699.4
  * clp_label                            (clp_label) <U2 's1' 's2' 's3'
  * species                              (species) <U2 's1' 's2' 's3'
    rate                                 (component) float64 -0.5001 ...
    lifetime                             (component) float64 -2.0 ... -10.0
  * to_species                           (to_species) <U2 's1' 's2' 's3'
  * from_species                         (from_species) <U2 's1' 's2' 's3'
Dimensions without coordinates: component, left_singular_value_index, right_singular_
↪value_index, singular_value_index
Data variables: (12/23)
    data                                 (time, spectral) float64 0.0127...
    data_left_singular_vectors           (time, left_singular_value_index) float64␣
↪...
    data_singular_values                 (singular_value_index) float64 ...
    data_right_singular_vectors          (spectral, right_singular_value_index)␣
↪float64 ...
    matrix                               (time, clp_label) float64 6.153...
    clp                                  (spectral, clp_label) float64 1...
    ...                                   ...
    decay_associated_spectra             (spectral, component) float64 2...
```

(continues on next page)

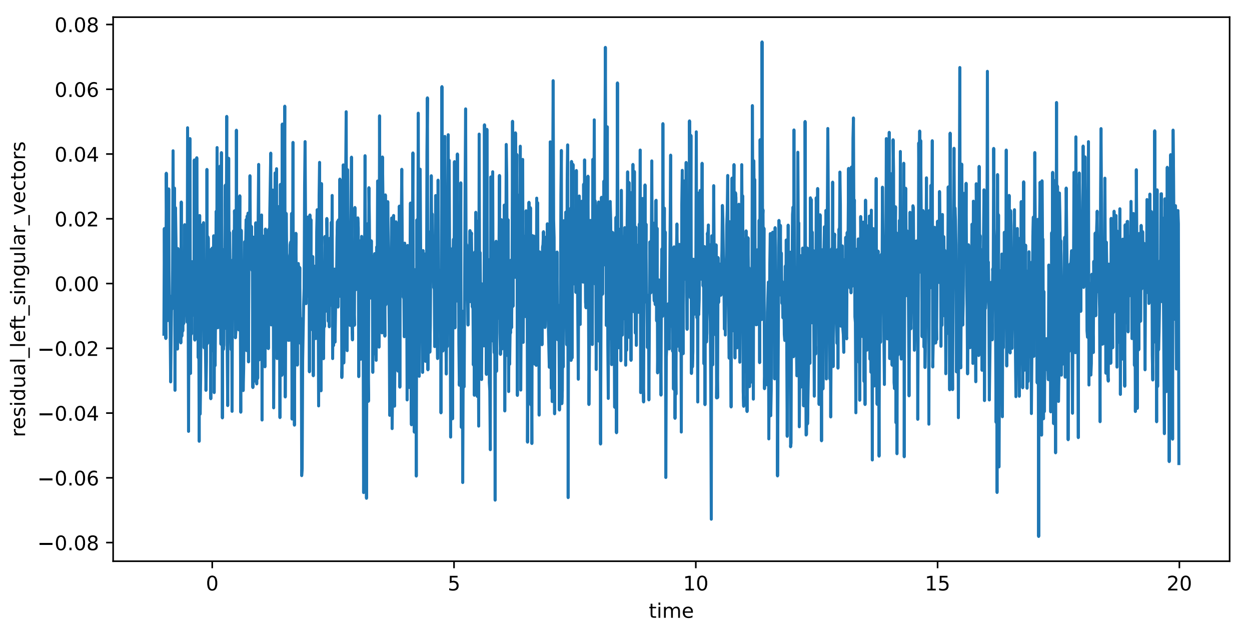
(continued from previous page)

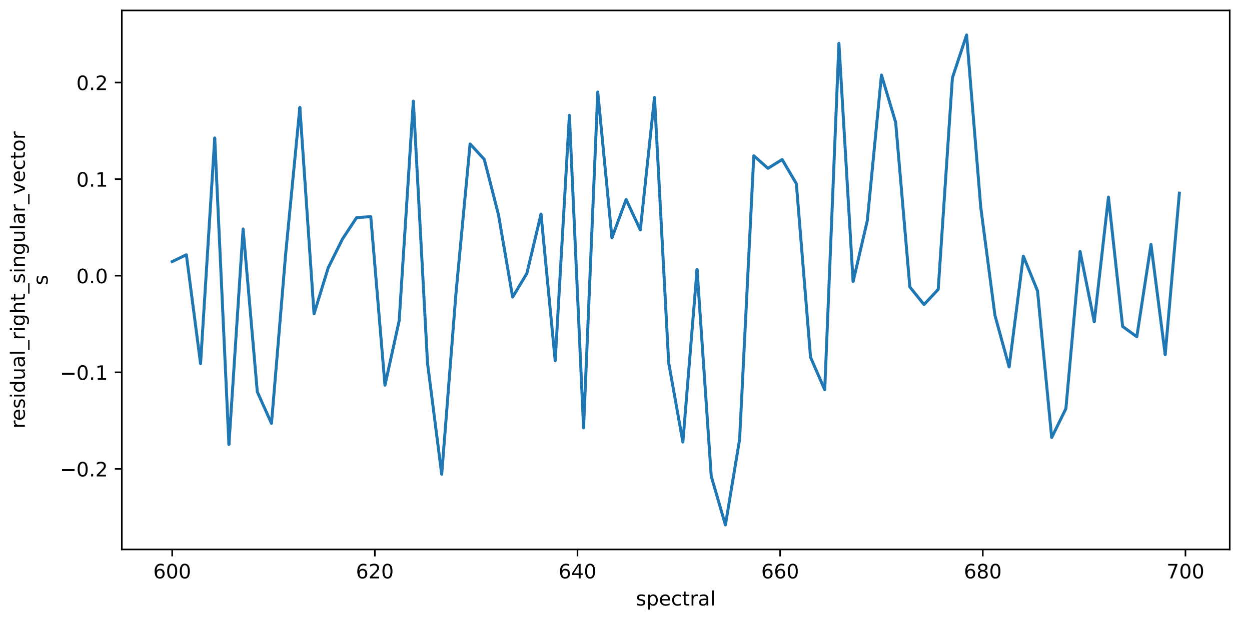
```
    a_matrix                                (component, species) float64 1...
    k_matrix                                (to_species, from_species) float64 ...
    k_matrix_reduced                        (to_species, from_species) float64 ...
    irf_center                              float64 0.3
    irf_width                               float64 0.1
Attributes:
    root_mean_square_error:          0.009997166315912881
    weighted_root_mean_square_error: 0.009997166315912881
```

3.6 Visualize the Result

The resulting data can be visualized the same way as the dataset. To judge the quality of the fit, you should look at first left and right singular vectors of the residual.

[19]:
```
residual_left = result_dataset.residual_left_singular_vectors.sel(left_singular_value_
↪index=0)
residual_right = result_dataset.residual_right_singular_vectors.sel(right_singular_value_
↪index=0)
residual_left.plot.line(x="time", aspect=2, size=5)
residual_right.plot.line(x="spectral", aspect=2, size=5);
```





Finally, you can save your result.

[20]:
```
save_dataset(result_dataset, "dataset1.nc")
```

CHAPTER

FOUR

CHANGELOG

4.1 0.4.0 (2021-06-25)

4.1.1 Features

- Add basic spectral model (#672)
- Add Channel/Wavelength dependent shift parameter to irf. (#673)
- Refactored Problem class into GroupedProblem and UngroupedProblem (#681)
- Plugin system was rewritten (#600, #665)
- Deprecation framework (#631)
- Better notebook integration (#689)

4.1.2 Bug fixes

- Fix excessive memory usage in `_create_svd` (#576)
- Fix several issues with KineticImage model (#612)
- Fix exception in sdt reader index calculation (#647)
- Avoid crash in result markdown printing when optimization fails (#630)
- ParameterNotFoundException doesn't prepend '.' if path is empty (#688)
- Ensure Parameter.label is str or None (#678)
- Properly scale StdError of estimated parameters with RMSE (#704)
- More robust covariance_matrix calculation (#706)
- `ParameterGroup.markdown()` independent parametergroups of order (#592)

4.1.3 Plugins

- `ProjectIo` ‘folder’/’legacy’ plugin to save results (#620)
- `Model` ‘spectral-model’ (#672)

4.1.4 Documentation

- User documentation is written in notebooks (#568)
- Documentation on how to write a `DataIo` plugin (#600)

4.1.5 Deprecations (due in 0.6.0)

- `glotaran.ParameterGroup` -> `glotaran.parameterParameterGroup`
- `glotaran.read_model_from_yaml` -> `glotaran.io.load_model(..., format_name="yaml_str")`
- `glotaran.read_model_from_yaml_file` -> `glotaran.io.load_model(..., format_name="yaml")`
- `glotaran.read_parameters_from_csv_file` -> `glotaran.io.load_parameters(..., format_name="csv")`
- `glotaran.read_parameters_from_yaml` -> `glotaran.io.load_parameters(..., format_name="yaml_str")`
- `glotaran.read_parameters_from_yaml_file` -> `glotaran.io.load_parameters(..., format_name="yaml")`
- `glotaran.io.read_data_file` -> `glotaran.io.load_dataset`
- `result.save` -> `glotaran.io.save_result(result, ..., format_name="legacy")`
- `result.get_dataset("<dataset_name>")` -> `result.data["<dataset_name>"]`
- `glotaran.analysis.result` -> `glotaran.project.result`
- `glotaran.analysis.scheme` -> `glotaran.project.scheme`
- `model.simulate` -> `glotaran.analysis.simulation.simulate(model, ...)`

4.2 0.3.3 (2021-03-18)

- Force recalculation of SVD attributes in `scheme._prepare_data` (#597)
- Remove unneeded check in `spectral_penalties._get_area` Fixes (#598)
- Added python 3.9 support (#450)

4.3 0.3.2 (2021-02-28)

- Re-release of version 0.3.1 due to packaging issue

4.4 0.3.1 (2021-02-28)

- Added compatibility for numpy 1.20 and raised minimum required numpy version to 1.20 (#555)
- Fixed excessive memory consumption in result creation due to full SVD computation (#574)
- Added feature parameter history (#557)
- Moved setup logic to `setup.cfg` (#560)

4.5 0.3.0 (2021-02-11)

- Significant code refactor with small API changes to parameter relation specification (see docs)
- Replaced lmfit with scipy.optimize

4.6 0.2.0 (2020-12-02)

- Large refactor with significant improvements but also small API changes (see docs)
- Removed doas plugin

4.7 0.1.0 (2020-07-14)

- Package was renamed to `pyglotaran` on PyPi

4.8 0.0.8 (2018-08-07)

- Changed `nan_policiy` to `omit`

4.9 0.0.7 (2018-08-07)

- Added support for multiple shapes per compartment.

4.10 0.0.6 (2018-08-07)

- First release on PyPI, support for Windows installs added.
- Pre-Alpha Development

CHAPTER

FIVE

AUTHORS

5.1 Development Lead

- Joern Weissenborn <joern.weissenborn@gmail.com>
- Joris Snellenburg <j.snellenburg@gmail.com>

5.2 Contributors

- Sebastian Weigand <s.weigand.phy@gmail.com>

5.3 Special Thanks

- Stefan Schuetz
- Sergey P. Laptenok

5.4 Supervision

- **dr. Ivo H.M. van Stokkum** <i.h.m.van.stokkum@vu.nl> (University profile)

5.5 Original publications

1. Joris J. Snellenburg, Sergey Laptenok, Ralf Seger, Katharine M. Mullen, Ivo H. M. van Stokkum. "Glotaran: A Java-Based Graphical User Interface for the R Package TIMP". Journal of Statistical Software (2012), Volume 49, Number 3, Pages: 1–22. URL https://dx.doi.org/10.18637/jss.v049.i03
2. Katharine M. Mullen, Ivo H. M. van Stokkum. "TIMP: An R Package for Modeling Multi-way Spectroscopic Measurements". Journal of Statistical Software (2007), Volume 18, Number 3, Pages 1-46, ISSN 1548-7660. URL https://dx.doi.org/10.18637/jss.v018.i03
3. Ivo H. M. van Stokkum, Delmar S. Larsen, Rienk van Grondelle, "Global and target analysis of time-resolved spectra". Biochimica et Biophysica Acta (BBA) - Bioenergetics (2004), Volume 1657, Issues 2–3, Pages 82-104, ISSN 0005-2728. URL https://doi.org/10.1016/j.bbabio.2004.04.011

CHAPTER SIX

OVERVIEW

CHAPTER SEVEN

DATA IO

CHAPTER

EIGHT

PLOTTING

CHAPTER NINE

MODELLING

CHAPTER TEN

PARAMETER

CHAPTER ELEVEN

OPTIMIZING

CHAPTER

TWELVE

API DOCUMENTATION

The API Documentation for pyglotaran is automatically created from its docstrings.

glotaran	Glotaran package __init__.py

12.1 glotaran

Glotaran package __init__.py

Modules

glotaran.analysis	This package contains functions for model simulation and fitting.
glotaran.builtin	This package contains builtin plugins.
glotaran.cli	
glotaran.deprecation	Deprecation helpers and place to put deprecated implementations till removing.
glotaran.examples	
glotaran.io	Functions for data IO
glotaran.model	Glotaran Model Package
glotaran.parameter	
glotaran.plugin_system	Plugin system package containing all plugin related implementations.
glotaran.project	
glotaran.utils	Glotaran utility function/class package.

12.1.1 analysis

This package contains functions for model simulation and fitting.

Modules

glotaran.analysis.nnls	Functions for calculating conditionally linear parameters and residual with the non-negative least-squares method.
glotaran.analysis.optimize	
glotaran.analysis.problem	
glotaran.analysis.problem_grouped	
glotaran.analysis.problem_ungrouped	
glotaran.analysis.simulation	Functions for simulating a global analysis model.
glotaran.analysis.util	
glotaran.analysis.variable_projection	Functions for calculating conditionally linear parameters and residual with the variable projection method.

nnls

Functions for calculating conditionally linear parameters and residual with the non-negative least-squares method.

Functions

Summary

residual_nnls	Calculate the conditionally linear parameters and residual with the nnls method.

residual_nnls

glotaran.analysis.nnls.**residual_nnls**(*matrix: numpy.ndarray*, *data: numpy.ndarray*) → Tuple[List[str], numpy.ndarray]

Calculate the conditionally linear parameters and residual with the nnls method.

nnls stands for 'non-negative least-squares'.

Parameters

- **matrix** – The model matrix.
- **data** (*np.ndarray*) – The data to analyze.

optimize

Functions

Summary

optimize	
optimize_problem	

optimize

glotaran.analysis.optimize.**optimize**(*scheme: glotaran.project.scheme.Scheme*, *verbose: bool = True*) → *glotaran.project.result.Result*

optimize_problem

glotaran.analysis.optimize.**optimize_problem**(*problem: glotaran.analysis.problem.Problem*, *verbose: bool = True*) → *glotaran.project.result.Result*

problem

Classes

Summary

GroupedProblemDescriptor	
Problem	A Problem class
ProblemGroup	
UngroupedProblemDescriptor	

GroupedProblemDescriptor

class glotaran.analysis.problem.**GroupedProblemDescriptor**(*label*, *indices*, *axis*)

Bases: `tuple`

Create new instance of GroupedProblemDescriptor(label, indices, axis)

Attributes Summary

axis	Alias for field number 2
indices	Alias for field number 1
label	Alias for field number 0

axis

GroupedProblemDescriptor.**axis**: dict[str, np.ndarray]

Alias for field number 2

indices

GroupedProblemDescriptor.**indices**: dict[str, int]

Alias for field number 1

label

GroupedProblemDescriptor.**label**: str

Alias for field number 0

Methods Summary

count	Return number of occurrences of value.
index	Return first index of value.

count

GroupedProblemDescriptor.**count**(*value*, */*)

Return number of occurrences of value.

index

`GroupedProblemDescriptor.index(value, start=0, stop=sys.maxsize, /)`
Return first index of value.

Raises ValueError if the value is not present.

Methods Documentation

axis: dict[str, np.ndarray]
Alias for field number 2

count(*value*, /)
Return number of occurrences of value.

index(*value*, *start=0*, *stop=sys.maxsize*, /)
Return first index of value.

Raises ValueError if the value is not present.

indices: dict[str, int]
Alias for field number 1

label: str
Alias for field number 0

Problem

class `glotaran.analysis.problem.Problem`(*scheme:* glotaran.project.scheme.Scheme)
Bases: `object`

A Problem class

Initializes the Problem class from a scheme (`glotaran.analysis.scheme.Scheme`)

Args:

scheme (Scheme): An instance of `glotaran.analysis.scheme.Scheme` which defines your model, parameters, and data

Attributes Summary

additional_penalty
bag
clp_labels
clps
cost
data

continues on next page

Table 9 – continued from previous page

filled_dataset_descriptors	
full_penalty	
grouped	
groups	
index_dependent	
matrices	
model	Property providing access to the used model
parameter_history	
parameters	
reduced_clp_labels	
reduced_clps	
reduced_matrices	
residuals	
scheme	Property providing access to the used scheme
weighted_residuals	

additional_penalty

Problem.**additional_penalty**

bag

Problem.**bag**

clp_labels

Problem.**clp_labels**

clps

Problem.**clps**

cost

Problem.**cost**

data

Problem.**data**

filled_dataset_descriptors

Problem.**filled_dataset_descriptors**

full_penalty

Problem.**full_penalty**

grouped

Problem.**grouped**

groups

Problem.**groups**

index_dependent

Problem.**index_dependent**

matrices

`Problem.matrices`

model

`Problem.model`

Property providing access to the used model

The model is a subclass of `glotaran.model.Model` decorated with the *@model* decorator `glotaran.model.model_decorator.model` For an example implementation see e.g. *glotaran.builtin.models.kinetic_spectrum*

Returns:

Model: A subclass of `glotaran.model.Model` The model must be decorated with the *@model* decorator `glotaran.model.model_decorator.model`

parameter_history

`Problem.parameter_history`

parameters

`Problem.parameters`

reduced_clp_labels

`Problem.reduced_clp_labels`

reduced_clps

`Problem.reduced_clps`

reduced_matrices

`Problem.reduced_matrices`

residuals

`Problem.residuals`

scheme

Problem.**scheme**

Property providing access to the used scheme

Returns:

Scheme: An instance of `glotaran.analysis.scheme.Scheme` Provides access to data, model, parameters and optimization arguments.

weighted_residuals

Problem.**weighted_residuals**

Methods Summary

calculate_additional_penalty	Calculates additional penalties by calling the model.additional_penalty function.
calculate_index_dependent_matrices	Calculates the index dependent model matrices.
calculate_index_dependent_residual	Calculates the index dependent residuals.
calculate_index_independent_matrices	Calculates the index independent model matrices.
calculate_index_independent_residual	Calculates the index independent residuals.
calculate_matrices	
calculate_residual	
create_index_dependent_result_dataset	Creates a result datasets for index dependent matrices.
create_index_independent_result_dataset	Creates a result datasets for index independent matrices.
create_result_data	
create_result_dataset	
init_bag	Initializes a problem bag.
reset	Resets all results and *DatasetDescriptors*.
save_parameters_for_history	

calculate_additional_penalty

`Problem.calculate_additional_penalty()` → np.ndarray | dict[str, np.ndarray]

Calculates additional penalties by calling the model.additional_penalty function.

calculate_index_dependent_matrices

`Problem.calculate_index_dependent_matrices()` → tuple[dict[str, list[list[str]]], dict[str, list[np.ndarray]], dict[str, list[str]], dict[str, list[np.ndarray]]]

Calculates the index dependent model matrices.

calculate_index_dependent_residual

`Problem.calculate_index_dependent_residual()` → tuple[dict[str, list[np.ndarray]], dict[str, list[np.ndarray]], dict[str, list[np.ndarray]], dict[str, list[np.ndarray]]]

Calculates the index dependent residuals.

calculate_index_independent_matrices

`Problem.calculate_index_independent_matrices()` → tuple[dict[str, list[str]], dict[str, np.ndarray], dict[str, list[str]], dict[str, np.ndarray]]

Calculates the index independent model matrices.

calculate_index_independent_residual

`Problem.calculate_index_independent_residual()` → tuple[dict[str, list[np.ndarray]], dict[str, list[np.ndarray]], dict[str, list[np.ndarray]], dict[str, list[np.ndarray]]]

Calculates the index independent residuals.

calculate_matrices

`Problem.calculate_matrices()`

calculate_residual

`Problem.calculate_residual()`

create_index_dependent_result_dataset

`Problem.create_index_dependent_result_dataset`(*label: str*, *dataset: xarray.core.dataset.Dataset*) → xarray.core.dataset.Dataset

Creates a result datasets for index dependent matrices.

create_index_independent_result_dataset

`Problem.create_index_independent_result_dataset`(*label: str*, *dataset: xarray.core.dataset.Dataset*) → xarray.core.dataset.Dataset

Creates a result datasets for index independent matrices.

create_result_data

`Problem.create_result_data`(*copy: bool = True*, *history_index: int | None = None*) → dict[str, xr.Dataset]

create_result_dataset

`Problem.create_result_dataset`(*label: str*, *copy: bool = True*) → xarray.core.dataset.Dataset

init_bag

`Problem.init_bag()`

Initializes a problem bag.

reset

`Problem.reset()`

Resets all results and *DatasetDescriptors*. Use after updating parameters.

save_parameters_for_history

`Problem.save_parameters_for_history()`

Methods Documentation

property additional_penalty: dict[str, list[float]]

property bag: UngroupedBag | GroupedBag

calculate_additional_penalty() → np.ndarray | dict[str, np.ndarray]
Calculates additional penalties by calling the model.additional_penalty function.

calculate_index_dependent_matrices() → tuple[dict[str, list[list[str]]], dict[str, list[np.ndarray]], dict[str, list[str]], dict[str, list[np.ndarray]]]
Calculates the index dependent model matrices.

calculate_index_dependent_residual() → tuple[dict[str, list[np.ndarray]], dict[str, list[np.ndarray]], dict[str, list[np.ndarray]], dict[str, list[np.ndarray]]]
Calculates the index dependent residuals.

calculate_index_independent_matrices() → tuple[dict[str, list[str]], dict[str, np.ndarray], dict[str, list[str]], dict[str, np.ndarray]]
Calculates the index independent model matrices.

calculate_index_independent_residual() → tuple[dict[str, list[np.ndarray]], dict[str, list[np.ndarray]], dict[str, list[np.ndarray]], dict[str, list[np.ndarray]]]
Calculates the index independent residuals.

calculate_matrices()

calculate_residual()

property clp_labels: dict[str, list[str] | list[list[str]]]

property clps: dict[str, list[np.ndarray]]

property cost: float

create_index_dependent_result_dataset(*label: str*, *dataset: xarray.core.dataset.Dataset*) → xarray.core.dataset.Dataset
Creates a result datasets for index dependent matrices.

create_index_independent_result_dataset(*label: str*, *dataset: xarray.core.dataset.Dataset*) → xarray.core.dataset.Dataset
Creates a result datasets for index independent matrices.

create_result_data(*copy: bool = True*, *history_index: int | None = None*) → dict[str, xr.Dataset]

create_result_dataset(*label: str*, *copy: bool = True*) → xarray.core.dataset.Dataset

property data: dict[str, xr.Dataset]

property filled_dataset_descriptors: dict[str, DatasetDescriptor]

property full_penalty: numpy.ndarray

property grouped: bool

property groups: dict[str, list[str]]

property index_dependent: bool

init_bag()
Initializes a problem bag.

property matrices: dict[str, np.ndarray | list[np.ndarray]]

property model: glotaran.model.base_model.Model
Property providing access to the used model

The model is a subclass of `glotaran.model.Model` decorated with the *@model* decorator `glotaran.model.model_decorator.model` For an example implementation see e.g. *glotaran.builtin.models.kinetic_spectrum*

Returns:

Model: A subclass of `glotaran.model.Model` The model must be decorated with the *@model* decorator `glotaran.model.model_decorator.model`

property parameter_history: list[ParameterGroup]

property parameters: glotaran.parameter.parameter_group.ParameterGroup

property reduced_clp_labels: dict[str, list[str] | list[list[str]]]

property reduced_clps: dict[str, list[np.ndarray]]

property reduced_matrices: dict[str, np.ndarray] | dict[str, list[np.ndarray]] | list[np.ndarray]

reset()
Resets all results and *DatasetDescriptors*. Use after updating parameters.

property residuals: dict[str, list[np.ndarray]]

save_parameters_for_history()

property scheme: glotaran.project.scheme.Scheme
Property providing access to the used scheme

Returns:

Scheme: An instance of `glotaran.analysis.scheme.Scheme` Provides access to data, model, parameters and optimization arguments.

property weighted_residuals: dict[str, list[np.ndarray]]

ProblemGroup

class glotaran.analysis.problem.**ProblemGroup**(*data*, *weight*, *has_scaling*, *group*, *data_sizes*, *descriptor*)

Bases: tuple

Create new instance of ProblemGroup(data, weight, has_scaling, group, data_sizes, descriptor)

Attributes Summary

data	Alias for field number 0
data_sizes	Holds the sizes of the concatenated datasets.
descriptor	Alias for field number 5
group	The concatenated labels of the involved datasets.
has_scaling	Indicates if at least one dataset in the group needs scaling.
weight	Alias for field number 1

data

ProblemGroup.**data: np.ndarray**

Alias for field number 0

data_sizes

ProblemGroup.**data_sizes: list[int]**

Holds the sizes of the concatenated datasets.

descriptor

ProblemGroup.**descriptor: list[GroupedProblemDescriptor]**

Alias for field number 5

group

ProblemGroup.**group: str**

The concatenated labels of the involved datasets.

has_scaling

ProblemGroup.**has_scaling: bool**

Indicates if at least one dataset in the group needs scaling.

weight

ProblemGroup.**weight: np.ndarray**

Alias for field number 1

Methods Summary

count	Return number of occurrences of value.
index	Return first index of value.

count

ProblemGroup.**count**(*value*, */*)

Return number of occurrences of value.

index

ProblemGroup.**index**(*value*, *start=0*, *stop=sys.maxsize*, */*)

Return first index of value.

Raises ValueError if the value is not present.

Methods Documentation

count(*value*, */*)

Return number of occurrences of value.

data: np.ndarray

Alias for field number 0

data_sizes: list[int]

Holds the sizes of the concatenated datasets.

descriptor: list[*GroupedProblemDescriptor*]

Alias for field number 5

group: str

The concatenated labels of the involved datasets.

has_scaling: bool

Indicates if at least one dataset in the group needs scaling.

index(*value*, *start=0*, *stop=sys.maxsize*, */*)

Return first index of value.

Raises ValueError if the value is not present.

weight: np.ndarray
Alias for field number 1

UngroupedProblemDescriptor

class glotaran.analysis.problem.**UngroupedProblemDescriptor**(*dataset*, *data*, *model_axis*, *global_axis*, *weight*)

Bases: `tuple`

Create new instance of UngroupedProblemDescriptor(dataset, data, model_axis, global_axis, weight)

Attributes Summary

data	Alias for field number 1
dataset	Alias for field number 0
global_axis	Alias for field number 3
model_axis	Alias for field number 2
weight	Alias for field number 4

data

UngroupedProblemDescriptor.**data: xarray.core.dataarray.DataArray**
Alias for field number 1

dataset

UngroupedProblemDescriptor.**dataset:** *glotaran.model.dataset_descriptor.DatasetDescriptor*
Alias for field number 0

global_axis

UngroupedProblemDescriptor.**global_axis: numpy.ndarray**
Alias for field number 3

model_axis

UngroupedProblemDescriptor.**model_axis: numpy.ndarray**
Alias for field number 2

weight

UngroupedProblemDescriptor.**weight: xarray.core.dataarray.DataArray**
Alias for field number 4

Methods Summary

count	Return number of occurrences of value.
index	Return first index of value.

count

UngroupedProblemDescriptor.**count**(*value*, */*)
Return number of occurrences of value.

index

UngroupedProblemDescriptor.**index**(*value*, *start=0*, *stop=sys.maxsize*, */*)
Return first index of value.

Raises ValueError if the value is not present.

Methods Documentation

count(*value*, */*)
Return number of occurrences of value.

data: xarray.core.dataarray.DataArray
Alias for field number 1

dataset: *glotaran.model.dataset_descriptor.DatasetDescriptor*
Alias for field number 0

global_axis: numpy.ndarray
Alias for field number 3

index(*value*, *start=0*, *stop=sys.maxsize*, */*)
Return first index of value.

Raises ValueError if the value is not present.

model_axis: numpy.ndarray
Alias for field number 2

weight: xarray.core.dataarray.DataArray
Alias for field number 4

Exceptions

Exception Summary

ParameterError

ParameterError

exception glotaran.analysis.problem.**ParameterError**

problem_grouped

Classes

Summary

GroupedProblem	Represents a problem where the data is grouped.

GroupedProblem

class glotaran.analysis.problem_grouped.**GroupedProblem**(*scheme: glotaran.project.scheme.Scheme*)

Bases: *glotaran.analysis.problem.Problem*

Represents a problem where the data is grouped.

Initializes the Problem class from a scheme (`glotaran.analysis.scheme.Scheme`)

Args:

scheme (Scheme): An instance of `glotaran.analysis.scheme.Scheme` which defines your model, parameters, and data

Attributes Summary

additional_penalty
bag
clp_labels
clps
cost
data

continues on next page

Table 17 – continued from previous page

filled_dataset_descriptors	
full_penalty	
grouped	
groups	
index_dependent	
matrices	
model	Property providing access to the used model
parameter_history	
parameters	
reduced_clp_labels	
reduced_clps	
reduced_matrices	
residuals	
scheme	Property providing access to the used scheme
weighted_residuals	

additional_penalty

GroupedProblem.**additional_penalty**

bag

GroupedProblem.**bag**

clp_labels

GroupedProblem.**clp_labels**

clps

GroupedProblem.**clps**

cost

GroupedProblem.**cost**

data

GroupedProblem.**data**

filled_dataset_descriptors

GroupedProblem.**filled_dataset_descriptors**

full_penalty

GroupedProblem.**full_penalty**

grouped

GroupedProblem.**grouped**

groups

GroupedProblem.**groups**

index_dependent

GroupedProblem.**index_dependent**

matrices

`GroupedProblem.matrices`

model

`GroupedProblem.model`

Property providing access to the used model

The model is a subclass of `glotaran.model.Model` decorated with the *@model* decorator `glotaran.model.model_decorator.model` For an example implementation see e.g. *glotaran.builtin.models.kinetic_spectrum*

Returns:

Model: A subclass of `glotaran.model.Model` The model must be decorated with the *@model* decorator `glotaran.model.model_decorator.model`

parameter_history

`GroupedProblem.parameter_history`

parameters

`GroupedProblem.parameters`

reduced_clp_labels

`GroupedProblem.reduced_clp_labels`

reduced_clps

`GroupedProblem.reduced_clps`

reduced_matrices

`GroupedProblem.reduced_matrices`

residuals

`GroupedProblem.residuals`

scheme

GroupedProblem.**scheme**

Property providing access to the used scheme

Returns:

Scheme: An instance of `glotaran.analysis.scheme.Scheme` Provides access to data, model, parameters and optimization arguments.

weighted_residuals

GroupedProblem.**weighted_residuals**

Methods Summary

calculate_additional_penalty	Calculates additional penalties by calling the model.additional_penalty function.
calculate_index_dependent_matrices	Calculates the index dependent model matrices.
calculate_index_dependent_residual	Calculates the index dependent residuals.
calculate_index_independent_matrices	Calculates the index independent model matrices.
calculate_index_independent_residual	Calculates the index independent residuals.
calculate_matrices	
calculate_residual	
create_index_dependent_result_dataset	Creates a result datasets for index dependent matrices.
create_index_independent_result_dataset	Creates a result datasets for index independent matrices.
create_result_data	
create_result_dataset	
init_bag	Initializes a grouped problem bag.
reset	Resets all results and *DatasetDescriptors*.
save_parameters_for_history	

calculate_additional_penalty

`GroupedProblem.calculate_additional_penalty()` → np.ndarray | dict[str, np.ndarray]

Calculates additional penalties by calling the model.additional_penalty function.

calculate_index_dependent_matrices

`GroupedProblem.calculate_index_dependent_matrices()` → tuple[dict[str, list[list[str]]], dict[str, list[np.ndarray]], list[list[str]], list[np.ndarray]]

Calculates the index dependent model matrices.

calculate_index_dependent_residual

`GroupedProblem.calculate_index_dependent_residual()` → tuple[list[np.ndarray], list[np.ndarray], list[np.ndarray], list[np.ndarray]]

Calculates the index dependent residuals.

calculate_index_independent_matrices

`GroupedProblem.calculate_index_independent_matrices()` → tuple[dict[str, list[str]], dict[str, np.ndarray], dict[str, LabelAndMatrix]]

Calculates the index independent model matrices.

calculate_index_independent_residual

`GroupedProblem.calculate_index_independent_residual()` → tuple[list[np.ndarray], list[np.ndarray], list[np.ndarray], list[np.ndarray]]

Calculates the index independent residuals.

calculate_matrices

`GroupedProblem.calculate_matrices()`

calculate_residual

GroupedProblem.**calculate_residual**()

create_index_dependent_result_dataset

GroupedProblem.**create_index_dependent_result_dataset**(*label: str*, *dataset: xarray.core.dataset.Dataset*) → xarray.core.dataset.Dataset

Creates a result datasets for index dependent matrices.

create_index_independent_result_dataset

GroupedProblem.**create_index_independent_result_dataset**(*label: str*, *dataset: xarray.core.dataset.Dataset*) → xarray.core.dataset.Dataset

Creates a result datasets for index independent matrices.

create_result_data

GroupedProblem.**create_result_data**(*copy: bool = True*, *history_index: int | None = None*) → dict[str, xr.Dataset]

create_result_dataset

GroupedProblem.**create_result_dataset**(*label: str*, *copy: bool = True*) → xarray.core.dataset.Dataset

init_bag

GroupedProblem.**init_bag**()

Initializes a grouped problem bag.

reset

GroupedProblem.**reset**()

Resets all results and *DatasetDescriptors*. Use after updating parameters.

save_parameters_for_history

`GroupedProblem.save_parameters_for_history()`

Methods Documentation

property additional_penalty: dict[str, list[float]]

property bag: UngroupedBag | GroupedBag

calculate_additional_penalty() → np.ndarray | dict[str, np.ndarray]
Calculates additional penalties by calling the model.additional_penalty function.

calculate_index_dependent_matrices() → tuple[dict[str, list[list[str]]], dict[str, list[np.ndarray]], list[list[str]], list[np.ndarray]]
Calculates the index dependent model matrices.

calculate_index_dependent_residual() → tuple[list[np.ndarray], list[np.ndarray], list[np.ndarray], list[np.ndarray]]
Calculates the index dependent residuals.

calculate_index_independent_matrices() → tuple[dict[str, list[str]], dict[str, np.ndarray], dict[str, LabelAndMatrix]]
Calculates the index independent model matrices.

calculate_index_independent_residual() → tuple[list[np.ndarray], list[np.ndarray], list[np.ndarray], list[np.ndarray]]
Calculates the index independent residuals.

calculate_matrices()

calculate_residual()

property clp_labels: dict[str, list[str] | list[list[str]]]

property clps: dict[str, list[np.ndarray]]

property cost: float

create_index_dependent_result_dataset(*label: str*, *dataset: xarray.core.dataset.Dataset*) → xarray.core.dataset.Dataset
Creates a result datasets for index dependent matrices.

create_index_independent_result_dataset(*label: str*, *dataset: xarray.core.dataset.Dataset*) → xarray.core.dataset.Dataset
Creates a result datasets for index independent matrices.

create_result_data(*copy: bool = True*, *history_index: int | None = None*) → dict[str, xr.Dataset]

create_result_dataset(*label: str*, *copy: bool = True*) → xarray.core.dataset.Dataset

property data: dict[str, xr.Dataset]

property filled_dataset_descriptors: dict[str, DatasetDescriptor]

property full_penalty: numpy.ndarray

property grouped: bool

property groups: dict[str, list[str]]

property index_dependent: bool

init_bag()

Initializes a grouped problem bag.

property matrices: dict[str, np.ndarray | list[np.ndarray]]

property model: glotaran.model.base_model.Model

Property providing access to the used model

The model is a subclass of `glotaran.model.Model` decorated with the *@model* decorator `glotaran.model.model_decorator.model` For an example implementation see e.g. *glotaran.builtin.models.kinetic_spectrum*

Returns:

Model: A subclass of `glotaran.model.Model` The model must be decorated with the *@model* decorator `glotaran.model.model_decorator.model`

property parameter_history: list[ParameterGroup]

property parameters: glotaran.parameter.parameter_group.ParameterGroup

property reduced_clp_labels: dict[str, list[str] | list[list[str]]]

property reduced_clps: dict[str, list[np.ndarray]]

property reduced_matrices: dict[str, np.ndarray] | dict[str, list[np.ndarray]] | list[np.ndarray]

reset()

Resets all results and *DatasetDescriptors*. Use after updating parameters.

property residuals: dict[str, list[np.ndarray]]

save_parameters_for_history()

property scheme: glotaran.project.scheme.Scheme

Property providing access to the used scheme

Returns:

Scheme: An instance of `glotaran.analysis.scheme.Scheme` Provides access to data, model, parameters and optimization arguments.

property weighted_residuals: dict[str, list[np.ndarray]]

problem_ungrouped

Classes

Summary

UngroupedProblem	Represents a problem where the data is not grouped.

UngroupedProblem

class glotaran.analysis.problem_ungrouped.**UngroupedProblem**(*scheme: glotaran.project.scheme.Scheme*)

Bases: *glotaran.analysis.problem.Problem*

Represents a problem where the data is not grouped.

Initializes the Problem class from a scheme (glotaran.analysis.scheme.Scheme)

Args:

scheme (Scheme): An instance of glotaran.analysis.scheme.Scheme which defines your model, parameters, and data

Attributes Summary

additional_penalty	
bag	
clp_labels	
clps	
cost	
data	
filled_dataset_descriptors	
full_penalty	
grouped	
groups	
index_dependent	
matrices	
model	Property providing access to the used model
parameter_history	
parameters	
reduced_clp_labels	
reduced_clps	
reduced_matrices	

continues on next page

Table 20 – continued from previous page

residuals	
scheme	Property providing access to the used scheme
weighted_residuals	

additional_penalty

UngroupedProblem.**additional_penalty**

bag

UngroupedProblem.**bag**

clp_labels

UngroupedProblem.**clp_labels**

clps

UngroupedProblem.**clps**

cost

UngroupedProblem.**cost**

data

UngroupedProblem.**data**

filled_dataset_descriptors

UngroupedProblem.**filled_dataset_descriptors**

full_penalty

UngroupedProblem.**full_penalty**

grouped

UngroupedProblem.**grouped**

groups

UngroupedProblem.**groups**

index_dependent

UngroupedProblem.**index_dependent**

matrices

UngroupedProblem.**matrices**

model

UngroupedProblem.**model**

Property providing access to the used model

The model is a subclass of `glotaran.model.Model` decorated with the *@model* decorator `glotaran.model.model_decorator.model` For an example implementation see e.g. *glotaran.builtin.models.kinetic_spectrum*

Returns:

Model: A subclass of `glotaran.model.Model` The model must be decorated with the *@model* decorator `glotaran.model.model_decorator.model`

parameter_history

UngroupedProblem.**parameter_history**

parameters

UngroupedProblem.**parameters**

reduced_clp_labels

UngroupedProblem.**reduced_clp_labels**

reduced_clps

UngroupedProblem.**reduced_clps**

reduced_matrices

UngroupedProblem.**reduced_matrices**

residuals

UngroupedProblem.**residuals**

scheme

UngroupedProblem.**scheme**

Property providing access to the used scheme

Returns:

Scheme: An instance of `glotaran.analysis.scheme.Scheme` Provides access to data, model, parameters and optimization arguments.

weighted_residuals

UngroupedProblem.**weighted_residuals**

Methods Summary

calculate_additional_penalty	Calculates additional penalties by calling the model.additional_penalty function.
calculate_index_dependent_matrices	Calculates the index dependent model matrices.
calculate_index_dependent_residual	Calculates the index dependent residuals.
calculate_index_independent_matrices	Calculates the index independent model matrices.
calculate_index_independent_residual	Calculates the index independent residuals.
calculate_matrices	
calculate_residual	
create_index_dependent_result_dataset	Creates a result datasets for index dependent matrices.
create_index_independent_result_dataset	Creates a result datasets for index independent matrices.
create_result_data	
create_result_dataset	
init_bag	Initializes an ungrouped problem bag.

continues on next page

Table 21 – continued from previous page

reset	Resets all results and *DatasetDescriptors*.
save_parameters_for_history	

calculate_additional_penalty

UngroupedProblem.**calculate_additional_penalty**() → np.ndarray | dict[str, np.ndarray]

Calculates additional penalties by calling the model.additional_penalty function.

calculate_index_dependent_matrices

UngroupedProblem.**calculate_index_dependent_matrices**() → tuple[dict[str, list[list[str]]], dict[str, list[np.ndarray]], dict[str, list[str]], dict[str, list[np.ndarray]]]

Calculates the index dependent model matrices.

calculate_index_dependent_residual

UngroupedProblem.**calculate_index_dependent_residual**() → tuple[dict[str, list[np.ndarray]], dict[str, list[np.ndarray]], dict[str, list[np.ndarray]], dict[str, list[np.ndarray]]]

Calculates the index dependent residuals.

calculate_index_independent_matrices

UngroupedProblem.**calculate_index_independent_matrices**() → tuple[dict[str, list[str]], dict[str, np.ndarray], dict[str, list[str]], dict[str, np.ndarray]]

Calculates the index independent model matrices.

calculate_index_independent_residual

UngroupedProblem.**calculate_index_independent_residual**() → tuple[dict[str, list[np.ndarray]], dict[str, list[np.ndarray]], dict[str, list[np.ndarray]], dict[str, list[np.ndarray]]]

Calculates the index independent residuals.

calculate_matrices

`UngroupedProblem.calculate_matrices()`

calculate_residual

`UngroupedProblem.calculate_residual()`

create_index_dependent_result_dataset

`UngroupedProblem.create_index_dependent_result_dataset(`*label: str*, *dataset: xarray.core.dataset.Dataset*`)` → xarray.core.dataset.Dataset

Creates a result datasets for index dependent matrices.

create_index_independent_result_dataset

`UngroupedProblem.create_index_independent_result_dataset(`*label: str*, *dataset: xarray.core.dataset.Dataset*`)` → xarray.core.dataset.Dataset

Creates a result datasets for index independent matrices.

create_result_data

`UngroupedProblem.create_result_data(`*copy: bool = True*, *history_index: int | None = None*`)` → dict[str, xr.Dataset]

create_result_dataset

`UngroupedProblem.create_result_dataset(`*label: str*, *copy: bool = True*`)` → xarray.core.dataset.Dataset

init_bag

`UngroupedProblem.init_bag()`

Initializes an ungrouped problem bag.

reset

`UngroupedProblem.reset()`

Resets all results and *DatasetDescriptors*. Use after updating parameters.

save_parameters_for_history

`UngroupedProblem.save_parameters_for_history()`

Methods Documentation

property additional_penalty: dict[str, list[float]]

property bag: UngroupedBag | GroupedBag

calculate_additional_penalty() → np.ndarray | dict[str, np.ndarray]

Calculates additional penalties by calling the model.additional_penalty function.

calculate_index_dependent_matrices() → tuple[dict[str, list[list[str]]], dict[str, list[np.ndarray]], dict[str, list[str]], dict[str, list[np.ndarray]]]

Calculates the index dependent model matrices.

calculate_index_dependent_residual() → tuple[dict[str, list[np.ndarray]], dict[str, list[np.ndarray]], dict[str, list[np.ndarray]], dict[str, list[np.ndarray]]]

Calculates the index dependent residuals.

calculate_index_independent_matrices() → tuple[dict[str, list[str]], dict[str, np.ndarray], dict[str, list[str]], dict[str, np.ndarray]]

Calculates the index independent model matrices.

calculate_index_independent_residual() → tuple[dict[str, list[np.ndarray]], dict[str, list[np.ndarray]], dict[str, list[np.ndarray]], dict[str, list[np.ndarray]]]

Calculates the index independent residuals.

calculate_matrices()

calculate_residual()

property clp_labels: dict[str, list[str] | list[list[str]]]

property clps: dict[str, list[np.ndarray]]

property cost: float

create_index_dependent_result_dataset(*label: str*, *dataset: xarray.core.dataset.Dataset*) → xarray.core.dataset.Dataset

Creates a result datasets for index dependent matrices.

create_index_independent_result_dataset(*label: str*, *dataset: xarray.core.dataset.Dataset*) → xarray.core.dataset.Dataset

Creates a result datasets for index independent matrices.

create_result_data(*copy: bool = True*, *history_index: int | None = None*) → dict[str, xr.Dataset]

create_result_dataset(*label: str*, *copy: bool = True*) → xarray.core.dataset.Dataset

property data: dict[str, xr.Dataset]

property filled_dataset_descriptors: dict[str, DatasetDescriptor]

property full_penalty: numpy.ndarray

property grouped: bool

property groups: dict[str, list[str]]

property index_dependent: bool

init_bag()

Initializes an ungrouped problem bag.

property matrices: dict[str, np.ndarray | list[np.ndarray]]

property model: glotaran.model.base_model.Model

Property providing access to the used model

The model is a subclass of `glotaran.model.Model` decorated with the *@model* decorator `glotaran.model.model_decorator.model` For an example implementation see e.g. *glotaran.builtin.models.kinetic_spectrum*

Returns:

Model: A subclass of `glotaran.model.Model` The model must be decorated with the *@model* decorator `glotaran.model.model_decorator.model`

property parameter_history: list[ParameterGroup]

property parameters: glotaran.parameter.parameter_group.ParameterGroup

property reduced_clp_labels: dict[str, list[str] | list[list[str]]]

property reduced_clps: dict[str, list[np.ndarray]]

property reduced_matrices: dict[str, np.ndarray] | dict[str, list[np.ndarray]] | list[np.ndarray]

reset()

Resets all results and *DatasetDescriptors*. Use after updating parameters.

property residuals: dict[str, list[np.ndarray]]

save_parameters_for_history()

property scheme: glotaran.project.scheme.Scheme

Property providing access to the used scheme

Returns:

Scheme: An instance of `glotaran.analysis.scheme.Scheme` Provides access to data, model, parameters and optimization arguments.

property weighted_residuals: dict[str, list[np.ndarray]]

simulation

Functions for simulating a global analysis model.

Functions

Summary

simulate	Simulates a model.

simulate

glotaran.analysis.simulation.**simulate**(*model: Model*, *dataset: str*, *parameters: ParameterGroup*, *axes: dict[str, np.ndarray] = None*, *clp: np.ndarray | xr.DataArray = None*, *noise=False*, *noise_std_dev=1.0*, *noise_seed=None*)

Simulates a model.

Parameters

- **model** – The model to simulate.
- **parameter** – The parameters for the simulation.
- **dataset** – Label of the dataset to simulate
- **axes** – A dictionary with axes for simulation.
- **clp** – conditionally linear parameters. Will be used instead of *model.global_matrix* if given.
- **noise** – Add noise to the simulation.
- **noise_std_dev** – The standard deviation for noise simulation.
- **noise_seed** – The seed for the noise simulation.

util

Functions

Summary

calculate_matrix
combine_matrices
find_closest_index
find_overlap

continues on next page

Table 23 – continued from previous page

get_min_max_from_interval	
reduce_matrix	

calculate_matrix

glotaran.analysis.util.**calculate_matrix**(*model: Model*, *dataset_descriptor: DatasetDescriptor*, *indices: dict[str, int]*, *axis: dict[str, np.ndarray]*) → LabelAndMatrix

combine_matrices

glotaran.analysis.util.**combine_matrices**(*labels_and_matrices: list[LabelAndMatrix]*) → LabelAndMatrix

find_closest_index

glotaran.analysis.util.**find_closest_index**(*index: float*, *axis: numpy.ndarray*)

find_overlap

glotaran.analysis.util.**find_overlap**(*a*, *b*, *rtol=1e-05*, *atol=1e-08*)

get_min_max_from_interval

glotaran.analysis.util.**get_min_max_from_interval**(*interval*, *axis*)

reduce_matrix

glotaran.analysis.util.**reduce_matrix**(*model: Model*, *label: str*, *parameters: ParameterGroup*, *result: LabelAndMatrix*, *index: float | None*) → LabelAndMatrix

Classes

Summary

LabelAndMatrix

LabelAndMatrix

`class glotaran.analysis.util.LabelAndMatrix(clp_label, matrix)`

Bases: `tuple`

Create new instance of LabelAndMatrix(clp_label, matrix)

Attributes Summary

clp_label	Alias for field number 0
matrix	Alias for field number 1

clp_label

`LabelAndMatrix.clp_label: list[str]`

Alias for field number 0

matrix

`LabelAndMatrix.matrix: np.ndarray`

Alias for field number 1

Methods Summary

count	Return number of occurrences of value.
index	Return first index of value.

count

`LabelAndMatrix.count(value, /)`

Return number of occurrences of value.

index

LabelAndMatrix.**index**(*value*, *start=0*, *stop=sys.maxsize*, */*)

Return first index of value.

Raises ValueError if the value is not present.

Methods Documentation

clp_label: list[str]

Alias for field number 0

count(*value*, */*)

Return number of occurrences of value.

index(*value*, *start=0*, *stop=sys.maxsize*, */*)

Return first index of value.

Raises ValueError if the value is not present.

matrix: np.ndarray

Alias for field number 1

variable_projection

Functions for calculating conditionally linear parameters and residual with the variable projection method.

Functions

Summary

residual_variable_projection	Calculates the conditionally linear parameters and residual with the variable projection method.

residual_variable_projection

glotaran.analysis.variable_projection.**residual_variable_projection**(*matrix: numpy.ndarray*, *data: numpy.ndarray*) → Tuple[List[str], numpy.ndarray]

Calculates the conditionally linear parameters and residual with the variable projection method.

Parameters

- **matrix** – The model matrix.
- **data** (*np.ndarray*) – The data to analyze.

12.1.2 builtin

This package contains builtin plugins.

Modules

glotaran.builtin.io	
glotaran.builtin.models	Glotaran Models Package

io

Modules

glotaran.builtin.io.ascii	
glotaran.builtin.io.csv	
glotaran.builtin.io.folder	Plugin to dump pyglotaran object as files in a folder.
glotaran.builtin.io.netCDF	
glotaran.builtin.io.sdt	
glotaran.builtin.io.yml	

ascii

Modules

glotaran.builtin.io.ascii.wavelength_time_explicit_file

wavelength_time_explicit_file

Functions

Summary

get_data_file_format
get_interval_number

get_data_file_format

glotaran.builtin.io.ascii.wavelength_time_explicit_file.**get_data_file_format**(*line*)

get_interval_number

glotaran.builtin.io.ascii.wavelength_time_explicit_file.**get_interval_number**(*line*)

Classes

Summary

AsciiDataIo	Initialize a Data IO plugin with the name of the format.
DataFileType	An enumeration.
ExplicitFile	Abstract class representing either a time- or wavelength-explicit file.
TimeExplicitFile	Represents a time explicit file
WavelengthExplicitFile	Represents a wavelength explicit file

AsciiDataIo

class glotaran.builtin.io.ascii.wavelength_time_explicit_file.**AsciiDataIo**(*format_name: str*)

Bases: *glotaran.io.interface.DataIoInterface*

Initialize a Data IO plugin with the name of the format.

Parameters **format_name** (*str*) – Name of the supported format an instance uses.

Methods Summary

load_dataset	Reads an ascii file in wavelength- or time-explicit format.
save_dataset	Save data from xarray.Dataset to a file (**NOT IMPLEMENTED**).

load_dataset

AsciiDataIo.**load_dataset**(*file_name: str*) → xr.Dataset | xr.DataArray

Reads an ascii file in wavelength- or time-explicit format.

See [1] for documentation of this format.

Parameters **fname** (*str*) – Name of the ascii file.

Returns **dataset**

Return type xr.Dataset

Notes

save_dataset

AsciiDataIo.**save_dataset**(*dataset: xarray.core.dataarray.DataArray*, *file_name: str*, ***, *comment: str = ''*, *file_format: glotaran.builtin.io.ascii.wavelength_time_explicit_file.DataFileType = <DataFileType.time_explicit: 'Time explicit'>*, *number_format: str = '%.10e'*)

Save data from xarray.Dataset to a file (**NOT IMPLEMENTED**).

Parameters

- **dataset** (*xr.Dataset*) – Dataset to be saved to file.
- **file_name** (*str*) – File to write the data to.

Methods Documentation

load_dataset(*file_name: str*) → xr.Dataset | xr.DataArray

Reads an ascii file in wavelength- or time-explicit format.

See [1] for documentation of this format.

Parameters **fname** (*str*) – Name of the ascii file.

Returns **dataset**

Return type xr.Dataset

Notes

save_dataset(*dataset: xarray.core.dataarray.DataArray*, *file_name: str*, ***, *comment: str = ''*, *file_format: glotaran.builtin.io.ascii.wavelength_time_explicit_file.DataFileType = <DataFileType.time_explicit: 'Time explicit'>*, *number_format: str = '%.10e'*)

Save data from xarray.Dataset to a file (**NOT IMPLEMENTED**).

Parameters

- **dataset** (*xr.Dataset*) – Dataset to be saved to file.
- **file_name** (*str*) – File to write the data to.

DataFileType

class glotaran.builtin.io.ascii.wavelength_time_explicit_file.**DataFileType**(*value*)

Bases: enum.Enum

An enumeration.

Attributes Summary

time_explicit	
wavelength_explicit	

time_explicit

DataFileType.**time_explicit = 'Time explicit'**

wavelength_explicit

DataFileType.**wavelength_explicit = 'Wavelength explicit'**

time_explicit = 'Time explicit'

wavelength_explicit = 'Wavelength explicit'

ExplicitFile

class glotaran.builtin.io.ascii.wavelength_time_explicit_file.**ExplicitFile**(*filepath: Optional[str] = None, dataset: Optional[xarray.core.dataarray.D = None*)

Bases: object

Abstract class representing either a time- or wavelength-explicit file.

Methods Summary

dataset
get_data_row
get_explicit_axis
get_format_name
get_observations
get_secondary_axis
read
set_explicit_axis
write

dataset

ExplicitFile.**dataset**(*prepare: bool = True*) → xr.Dataset | xr.DataArray

get_data_row

ExplicitFile.**get_data_row**(*index*)

get_explicit_axis

ExplicitFile.**get_explicit_axis**()

get_format_name

ExplicitFile.**get_format_name**()

get_observations

ExplicitFile.**get_observations**(*index*)

get_secondary_axis

ExplicitFile.**get_secondary_axis**()

read

ExplicitFile.**read**(*prepare: bool = True*)

set_explicit_axis

ExplicitFile.**set_explicit_axis**(*axis*)

write

ExplicitFile.**write**(*overwrite=False, comment='', file_format=<DataFileType.time_explicit: 'Time explicit'>, number_format='%.10e'*)

Methods Documentation

dataset(*prepare: bool = True*) → xr.Dataset | xr.DataArray

get_data_row(*index*)

get_explicit_axis()

get_format_name()

get_observations(*index*)

get_secondary_axis()

read(*prepare: bool = True*)

set_explicit_axis(*axis*)

write(*overwrite=False*, *comment=''*, *file_format=<DataFileType.time_explicit: 'Time explicit'>*, *number_format='%.10e'*)

TimeExplicitFile

class glotaran.builtin.io.ascii.wavelength_time_explicit_file.**TimeExplicitFile**(*filepath: Optional[str] = None*, *dataset: Optional[xarray.core.dataar*

= None)

Bases: *glotaran.builtin.io.ascii.wavelength_time_explicit_file.ExplicitFile*

Represents a time explicit file

Methods Summary

add_data_row
dataset
get_data_row
get_explicit_axis
get_format_name
get_observations
get_secondary_axis
read
set_explicit_axis
write

add_data_row

TimeExplicitFile.**add_data_row**(*row*)

dataset

TimeExplicitFile.**dataset**(*prepare: bool = True*) → xr.Dataset | xr.DataArray

get_data_row

TimeExplicitFile.**get_data_row**(*index*)

get_explicit_axis

TimeExplicitFile.**get_explicit_axis**()

get_format_name

TimeExplicitFile.**get_format_name**()

get_observations

TimeExplicitFile.**get_observations**(*index*)

get_secondary_axis

TimeExplicitFile.**get_secondary_axis**()

read

TimeExplicitFile.**read**(*prepare: bool = True*)

set_explicit_axis

TimeExplicitFile.**set_explicit_axis**(*axies*)

write

TimeExplicitFile.**write**(*overwrite=False*, *comment=''*, *file_format=<DataFileType.time_explicit: 'Time explicit'>*, *number_format='%.10e'*)

Methods Documentation

add_data_row(*row*)

dataset(*prepare: bool = True*) → xr.Dataset | xr.DataArray

get_data_row(*index*)

get_explicit_axis()

get_format_name()

get_observations(*index*)

get_secondary_axis()

read(*prepare: bool = True*)

set_explicit_axis(*axies*)

write(*overwrite=False*, *comment=''*, *file_format=<DataFileType.time_explicit: 'Time explicit'>*, *number_format='%.10e'*)

WavelengthExplicitFile

class glotaran.builtin.io.ascii.wavelength_time_explicit_file.**WavelengthExplicitFile**(*filepath: Optional[str] = None, dataset: Optional[xarray.core*

= None)

Bases: *glotaran.builtin.io.ascii.wavelength_time_explicit_file.ExplicitFile*

Represents a wavelength explicit file

Methods Summary

add_data_row
dataset
get_data_row
get_explicit_axis
get_format_name
get_observations
get_secondary_axis
read
set_explicit_axis
times
wavelengths
write

add_data_row

`WavelengthExplicitFile.add_data_row(row)`

dataset

`WavelengthExplicitFile.dataset(prepare: bool = True)` → xr.Dataset | xr.DataArray

get_data_row

`WavelengthExplicitFile.get_data_row(index)`

get_explicit_axis

`WavelengthExplicitFile.get_explicit_axis()`

get_format_name

`WavelengthExplicitFile.get_format_name()`

get_observations

`WavelengthExplicitFile.get_observations(index)`

get_secondary_axis

`WavelengthExplicitFile.get_secondary_axis()`

read

`WavelengthExplicitFile.read(prepare: bool = True)`

set_explicit_axis

WavelengthExplicitFile.**set_explicit_axis**(*axis*)

times

WavelengthExplicitFile.**times**()

wavelengths

WavelengthExplicitFile.**wavelengths**()

write

WavelengthExplicitFile.**write**(*overwrite=False*, *comment=''*, *file_format=<DataFileType.time_explicit: 'Time explicit'>*, *number_format='%.10e'*)

Methods Documentation

add_data_row(*row*)

dataset(*prepare: bool = True*) → xr.Dataset | xr.DataArray

get_data_row(*index*)

get_explicit_axis()

get_format_name()

get_observations(*index*)

get_secondary_axis()

read(*prepare: bool = True*)

set_explicit_axis(*axis*)

times()

wavelengths()

write(*overwrite=False*, *comment=''*, *file_format=<DataFileType.time_explicit: 'Time explicit'>*, *number_format='%.10e'*)

csv

Modules

glotaran.builtin.io.csv.csv

csv

Classes

Summary

CsvProjectIo	Initialize a Project IO plugin with the name of the format.

CsvProjectIo

class glotaran.builtin.io.csv.csv.**CsvProjectIo**(*format_name: str*)

Bases: *glotaran.io.interface.ProjectIoInterface*

Initialize a Project IO plugin with the name of the format.

Parameters **format_name** (*str*) – Name of the supported format an instance uses.

Methods Summary

load_model	Create a Model instance from the specs defined in a file (**NOT IMPLEMENTED**).
load_parameters	Create a ParameterGroup instance from the specs defined in a file (**NOT IMPLEMENTED**).
load_result	Create a Result instance from the specs defined in a file (**NOT IMPLEMENTED**).
load_scheme	Create a Scheme instance from the specs defined in a file (**NOT IMPLEMENTED**).
save_model	Save a Model instance to a spec file (**NOT IMPLEMENTED**).
save_parameters	Save a ParameterGroup to a CSV file.

continues on next page

Table 40 – continued from previous page

save_result	Save a Result instance to a spec file (**NOT IMPLEMENTED**).
save_scheme	Save a Scheme instance to a spec file (**NOT IMPLEMENTED**).

load_model

CsvProjectIo.**load_model**(*file_name: str*) → Model

Create a Model instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **file_name** (*str*) – File containing the model specs.

Returns Model instance created from the file.

Return type *Model*

load_parameters

CsvProjectIo.**load_parameters**(*file_name: str*) → *glotaran.parameter.parameter_group.ParameterGroup*

Create a ParameterGroup instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **file_name** (*str*) – File containing the parameter specs.

Returns ParameterGroup instance created from the file.

Return type *ParameterGroup*

load_result

CsvProjectIo.**load_result**(*result_path: str*) → Result

Create a Result instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **result_path** (*str*) – Path containing the result data.

Returns Result instance created from the file.

Return type *Result*

load_scheme

CsvProjectIo.**load_scheme**(*file_name: str*) → Scheme

Create a Scheme instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **file_name** (*str*) – File containing the parameter specs.

Returns

- *Scheme* – Scheme instance created from the file.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

save_model

`CsvProjectIo.save_model(model: Model, file_name: str)`

Save a Model instance to a spec file (**NOT IMPLEMENTED**).

Parameters

- **model** (Model) – Model instance to save to specs file.
- **file_name** (*str*) – File to write the model specs to.

save_parameters

`CsvProjectIo.save_parameters(parameters: glotaran.parameter.parameter_group.ParameterGroup, file_name: str)`

Save a `ParameterGroup` to a CSV file.

save_result

`CsvProjectIo.save_result(result: Result, result_path: str)`

Save a Result instance to a spec file (**NOT IMPLEMENTED**).

Parameters

- **result** (Result) – Result instance to save to specs file.
- **result_path** (*str*) – Path to write the result data to.

save_scheme

`CsvProjectIo.save_scheme(scheme: Scheme, file_name: str)`

Save a Scheme instance to a spec file (**NOT IMPLEMENTED**).

Parameters

- **scheme** (Scheme) – Scheme instance to save to specs file.
- **file_name** (*str*) – File to write the scheme specs to.

Methods Documentation

`load_model(file_name: str) → Model`

Create a Model instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **file_name** (*str*) – File containing the model specs.

Returns Model instance created from the file.

Return type *Model*

`load_parameters(file_name: str) → glotaran.parameter.parameter_group.ParameterGroup`

Create a ParameterGroup instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **file_name** (*str*) – File containing the parameter specs.

Returns ParameterGroup instance created from the file.

Return type *ParameterGroup*

`load_result(result_path: str) → Result`

Create a Result instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **result_path** (*str*) – Path containing the result data.

Returns Result instance created from the file.

Return type *Result*

load_scheme(*file_name: str*) → Scheme

Create a Scheme instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **file_name** (*str*) – File containing the parameter specs.

Returns

- *Scheme* – Scheme instance created from the file.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

save_model(*model: Model*, *file_name: str*)

Save a Model instance to a spec file (**NOT IMPLEMENTED**).

Parameters

- **model** (Model) – Model instance to save to specs file.
- **file_name** (*str*) – File to write the model specs to.

save_parameters(*parameters:* glotaran.parameter.parameter_group.ParameterGroup, *file_name: str*)

Save a `ParameterGroup` to a CSV file.

save_result(*result: Result*, *result_path: str*)

Save a Result instance to a spec file (**NOT IMPLEMENTED**).

Parameters

- **result** (Result) – Result instance to save to specs file.
- **result_path** (*str*) – Path to write the result data to.

save_scheme(*scheme: Scheme*, *file_name: str*)

Save a Scheme instance to a spec file (**NOT IMPLEMENTED**).

Parameters

- **scheme** (Scheme) – Scheme instance to save to specs file.
- **file_name** (*str*) – File to write the scheme specs to.

folder

Plugin to dump pyglotaran object as files in a folder.

Modules

glotaran.builtin.io.folder.folder_plugin	Implementation of the folder Io plugin.

folder_plugin

Implementation of the folder Io plugin.

The current implementation is an exact copy of how `Result.save(path)` worked in glotaran 0.3.x and meant as an compatibility function.

Classes

Summary

FolderProjectIo	Project Io plugin to save result data to a folder.

FolderProjectIo

class glotaran.builtin.io.folder.folder_plugin.**FolderProjectIo**(*format_name: str*)

Bases: *glotaran.io.interface.ProjectIoInterface*

Project Io plugin to save result data to a folder.

There won't be a serialization of the Result object, but simply a markdown summary output and the important data saved to files.

Initialize a Project IO plugin with the name of the format.

Parameters **format_name** (*str*) – Name of the supported format an instance uses.

Methods Summary

load_model	Create a Model instance from the specs defined in a file (**NOT IMPLEMENTED**).
load_parameters	Create a ParameterGroup instance from the specs defined in a file (**NOT IMPLEMENTED**).
load_result	Create a Result instance from the specs defined in a file (**NOT IMPLEMENTED**).
load_scheme	Create a Scheme instance from the specs defined in a file (**NOT IMPLEMENTED**).
save_model	Save a Model instance to a spec file (**NOT IMPLEMENTED**).
save_parameters	Save a ParameterGroup instance to a spec file (**NOT IMPLEMENTED**).
save_result	Save the result to a given folder.
save_scheme	Save a Scheme instance to a spec file (**NOT IMPLEMENTED**).

load_model

FolderProjectIo.**load_model**(*file_name: str*) → Model

Create a Model instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **file_name** (*str*) – File containing the model specs.

Returns Model instance created from the file.

Return type *Model*

load_parameters

FolderProjectIo.**load_parameters**(*file_name: str*) → ParameterGroup

Create a ParameterGroup instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **file_name** (*str*) – File containing the parameter specs.

Returns ParameterGroup instance created from the file.

Return type *ParameterGroup*

load_result

FolderProjectIo.**load_result**(*result_path: str*) → Result

Create a Result instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **result_path** (*str*) – Path containing the result data.

Returns Result instance created from the file.

Return type *Result*

load_scheme

FolderProjectIo.**load_scheme**(*file_name: str*) → Scheme

Create a Scheme instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **file_name** (*str*) – File containing the parameter specs.

Returns

- *Scheme* – Scheme instance created from the file.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

save_model

FolderProjectIo.**save_model**(*model: Model*, *file_name: str*)

Save a Model instance to a spec file (**NOT IMPLEMENTED**).

Parameters

- **model** (Model) – Model instance to save to specs file.
- **file_name** (*str*) – File to write the model specs to.

save_parameters

FolderProjectIo.**save_parameters**(*parameters: ParameterGroup*, *file_name: str*)

Save a ParameterGroup instance to a spec file (**NOT IMPLEMENTED**).

Parameters

- **parameters** (ParameterGroup) – ParameterGroup instance to save to specs file.
- **file_name** (*str*) – File to write the parameter specs to.

save_result

`FolderProjectIo.save_result`(*result: Result*, *result_path: str*) → list[str]

Save the result to a given folder.

Returns a list with paths of all saved items. The following files are saved: * *result.md*: The result with the model formatted as markdown text. * *optimized_parameters.csv*: The optimized parameter as csv file. * *{dataset_label}.nc*: The result data for each dataset as NetCDF file.

Parameters
- **result** (`Result`) – Result instance to be saved.
- **result_path** (*str*) – The path to the folder in which to save the result.

Returns List of file paths which were created.

Return type list[str]

Raises `ValueError` – If `result_path` is a file.

save_scheme

`FolderProjectIo.save_scheme`(*scheme: Scheme*, *file_name: str*)

Save a Scheme instance to a spec file (**NOT IMPLEMENTED**).

Parameters
- **scheme** (`Scheme`) – Scheme instance to save to specs file.
- **file_name** (*str*) – File to write the scheme specs to.

Methods Documentation

`load_model`(*file_name: str*) → Model

Create a Model instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **file_name** (*str*) – File containing the model specs.

Returns Model instance created from the file.

Return type *Model*

`load_parameters`(*file_name: str*) → ParameterGroup

Create a ParameterGroup instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **file_name** (*str*) – File containing the parameter specs.

Returns ParameterGroup instance created from the file.

Return type *ParameterGroup*

`load_result`(*result_path: str*) → Result

Create a Result instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **result_path** (*str*) – Path containing the result data.

Returns Result instance created from the file.

Return type *Result*

`load_scheme`(*file_name: str*) → Scheme

Create a Scheme instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **file_name** (*str*) – File containing the parameter specs.

Returns
- *Scheme* – Scheme instance created from the file.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

`save_model`(*model: Model*, *file_name: str*)

Save a Model instance to a spec file (**NOT IMPLEMENTED**).

Parameters

- **model** (Model) – Model instance to save to specs file.
- **file_name** (*str*) – File to write the model specs to.

save_parameters(*parameters: ParameterGroup*, *file_name: str*)

Save a ParameterGroup instance to a spec file (**NOT IMPLEMENTED**).

Parameters

- **parameters** (ParameterGroup) – ParameterGroup instance to save to specs file.
- **file_name** (*str*) – File to write the parameter specs to.

save_result(*result: Result*, *result_path: str*) → list[str]

Save the result to a given folder.

Returns a list with paths of all saved items. The following files are saved: * *result.md*: The result with the model formatted as markdown text. * *optimized_parameters.csv*: The optimized parameter as csv file. * *{dataset_label}.nc*: The result data for each dataset as NetCDF file.

Parameters

- **result** (Result) – Result instance to be saved.
- **result_path** (*str*) – The path to the folder in which to save the result.

Returns List of file paths which were created.

Return type list[str]

Raises **ValueError** – If `result_path` is a file.

save_scheme(*scheme: Scheme*, *file_name: str*)

Save a Scheme instance to a spec file (**NOT IMPLEMENTED**).

Parameters

- **scheme** (Scheme) – Scheme instance to save to specs file.
- **file_name** (*str*) – File to write the scheme specs to.

netCDF

Modules

glotaran.builtin.io.netCDF.netCDF

netCDF

Classes

Summary

NetCDFDataIo	Initialize a Data IO plugin with the name of the format.

NetCDFDataIo

class `glotaran.builtin.io.netCDF.netCDF.`**`NetCDFDataIo`**(*format_name: str*)

Bases: `glotaran.io.interface.DataIoInterface`

Initialize a Data IO plugin with the name of the format.

Parameters **`format_name`** (`str`) – Name of the supported format an instance uses.

Methods Summary

`load_dataset`	Read data from a file to xarray.Dataset or xarray.DataArray (**NOT IMPLEMENTED**).
`save_dataset`	Save data from xarray.Dataset to a file (**NOT IMPLEMENTED**).

load_dataset

`NetCDFDataIo.`**`load_dataset`**(*file_name: str*) → xr.Dataset | xr.DataArray

Read data from a file to xarray.Dataset or xarray.DataArray (**NOT IMPLEMENTED**).

Parameters **`file_name`** (`str`) – File containing the data.

Returns Data loaded from the file.

Return type xr.Dataset|xr.DataArray

save_dataset

`NetCDFDataIo.`**`save_dataset`**(*dataset: xarray.core.dataset.Dataset*, *file_name: str*, *, *saving_options: glotaran.project.scheme.SavingOptions = SavingOptions(level='full', data_filter=None, data_format='nc', parameter_format='csv', report=True)*)

Save data from xarray.Dataset to a file (**NOT IMPLEMENTED**).

Parameters

- **`dataset`** (*`xr.Dataset`*) – Dataset to be saved to file.
- **`file_name`** (`str`) – File to write the data to.

Methods Documentation

`load_dataset`(*file_name: str*) → xr.Dataset | xr.DataArray

Read data from a file to xarray.Dataset or xarray.DataArray (**NOT IMPLEMENTED**).

Parameters **`file_name`** (`str`) – File containing the data.

Returns Data loaded from the file.

Return type xr.Dataset|xr.DataArray

`save_dataset`(*dataset: xarray.core.dataset.Dataset*, *file_name: str*, *, *saving_options: glotaran.project.scheme.SavingOptions = SavingOptions(level='full', data_filter=None, data_format='nc', parameter_format='csv', report=True)*)

Save data from xarray.Dataset to a file (**NOT IMPLEMENTED**).

Parameters

- **`dataset`** (*`xr.Dataset`*) – Dataset to be saved to file.
- **`file_name`** (`str`) – File to write the data to.

sdt

Modules

glotaran.builtin.io.sdt.sdt_file_reader	Glotarans module to read files

sdt_file_reader

Glotarans module to read files

Classes

Summary

SdtDataIo	Initialize a Data IO plugin with the name of the format.

SdtDataIo

class glotaran.builtin.io.sdt.sdt_file_reader.**SdtDataIo**(*format_name: str*)

Bases: *glotaran.io.interface.DataIoInterface*

Initialize a Data IO plugin with the name of the format.

Parameters **format_name** (*str*) – Name of the supported format an instance uses.

Methods Summary

load_dataset	Reads a **.sdt* file and returns a pd.DataFrame (*return_dataframe==True*), a SpectralTemporalDataset (*type_of_data=='st'*) or a FLIMDataset (*type_of_data=='flim'*).
save_dataset	Save data from xarray.Dataset to a file (**NOT IMPLEMENTED**).

load_dataset

SdtDataIo.**load_dataset**(*file_name: str*, ***, *index: np.ndarray | None = None*, *flim: bool = False*, *dataset_index: int | None = None*, *swap_axis: bool = False*, *orig_time_axis_index: int = 2*) → xr.Dataset

Reads a **.sdt* file and returns a pd.DataFrame (*return_dataframe==True*), a SpectralTemporalDataset (*type_of_data=='st'*) or a FLIMDataset (*type_of_data=='flim'*).

Parameters

- **file_name** (*str*) – Path to the sdt file which should be read.
- **index** (*list, np.ndarray*) – This is only needed if *type_of_data=="st"*, since **.sdt* files, which only contain spectral temporal data, lack the spectral information.

Thus for the spectral axis data need to be given by the user.
- **flim** – Set true if reading a result from a FLIM measurement.
- **dataset_index** (*int: default 0*) – If the **.sdt* file contains multiple datasets the index will used to select the wanted one
- **swap_axis** (*bool, default False*) – Flag to switch a wavelength explicit *input_df* to time explicit *input_df*, before generating the SpectralTemporalDataset.
- **orig_time_axis_index** (*int*) – Index of the axis which corresponds to the time axis. I.e. for data of shape (64, 64, 256), which are a 64x64 pixel map with 256 time steps, orig_time_axis_index=2.

Raises IndexError: – If the length of the index array is incompatible with the data.

save_dataset

SdtDataIo.**save_dataset**(*dataset: xr.Dataset | xr.DataArray*, *file_name: str*)

Save data from xarray.Dataset to a file (**NOT IMPLEMENTED**).

Parameters
- **dataset** (*xr.Dataset*) – Dataset to be saved to file.
- **file_name** (*str*) – File to write the data to.

Methods Documentation

load_dataset(*file_name: str*, ***, *index: np.ndarray | None = None*, *flim: bool = False*, *dataset_index: int | None = None*, *swap_axis: bool = False*, *orig_time_axis_index: int = 2*) → xr.Dataset

Reads a **.sdt* file and returns a pd.DataFrame (*return_dataframe==True*), a SpectralTemporalDataset (*type_of_data=='st'*) or a FLIMDataset (*type_of_data=='flim'*).

Parameters
- **file_name** (*str*) – Path to the sdt file which should be read.
- **index** (*list, np.ndarray*) – This is only needed if *type_of_data=="st"*, since **.sdt* files, which only contain spectral temporal data, lack the spectral information. Thus for the spectral axis data need to be given by the user.
- **flim** – Set true if reading a result from a FLIM measurement.
- **dataset_index** (*int: default 0*) – If the **.sdt* file contains multiple datasets the index will used to select the wanted one
- **swap_axis** (*bool, default False*) – Flag to switch a wavelength explicit *input_df* to time explicit *input_df*, before generating the SpectralTemporalDataset.
- **orig_time_axis_index** (*int*) – Index of the axis which corresponds to the time axis. I.e. for data of shape (64, 64, 256), which are a 64x64 pixel map with 256 time steps, orig_time_axis_index=2.

Raises IndexError: – If the length of the index array is incompatible with the data.

save_dataset(*dataset: xr.Dataset | xr.DataArray*, *file_name: str*)

Save data from xarray.Dataset to a file (**NOT IMPLEMENTED**).

Parameters
- **dataset** (*xr.Dataset*) – Dataset to be saved to file.
- **file_name** (*str*) – File to write the data to.

yml

Modules

glotaran.builtin.io.yml.sanatize
glotaran.builtin.io.yml.yml

sanatize

Functions

Summary

list_string_to_tuple	Converts a list of strings (representing tuples) to a list of tuples
sanitize_dict_keys	Sanitize the stringified tuple dict keys in a yaml parsed dict
sanitize_dict_values	Sanitizes a dict with broken tuples inside modifying it in-place Broken tuples are tuples that are turned into strings by the yaml parser.
sanitize_list_with_broken_tuples	Sanitize a list with 'broken' tuples
sanitize_yaml	Sanitize a yaml-returned dict for key or (list) values containing tuples
string_to_tuple	[summary]

list_string_to_tuple

glotaran.builtin.io.yml.sanatize.**list_string_to_tuple**(*a_list: List[str]*) → List[Union[str, float]]

Converts a list of strings (representing tuples) to a list of tuples

Parameters **a_list** (*List[str]*) – A list of strings, some of them representing (numbered) tuples

Returns A list of the (numbered) tuples represted by the incoming a_list

Return type List[Union[float, str]]

sanitize_dict_keys

`glotaran.builtin.io.yml.sanatize.sanitize_dict_keys(d: dict) → dict`

Sanitize the stringified tuple dict keys in a yaml parsed dict

Keys representing a tuple, e.g. '(s1, s2)' are converted to a tuple of strings e.g. ('s1', 's2')

Parameters **d** (*dict*) – A dict containing tuple-like string keys

Returns A dict with tuple-like string keys converted to tuple keys

Return type dict

sanitize_dict_values

`glotaran.builtin.io.yml.sanatize.sanitize_dict_values(d: dict)`

Sanitizes a dict with broken tuples inside modifying it in-place Broken tuples are tuples that are turned into strings by the yaml parser. This functions calls *sanitize_list_with_broken_tuples* to glue the broken strings together and then calls list_to_tuple to turn the list with tuple strings back to number tuples.

Args: d (dict): A (complex) dict containing (possibly nested) values of broken tuple strings

sanitize_list_with_broken_tuples

`glotaran.builtin.io.yml.sanatize.sanitize_list_with_broken_tuples(mangled_list: List[Union[str, float]]) → List[str]`

Sanitize a list with 'broken' tuples

A list of broken tuples as returned by yaml when parsing tuples. e.g parsing the list of tuples [(3,100), (4,200)] results in a list of str ['(3', '100)', '(4', '200)'] which can be restored to a list with the tuples restored as strings ['(3, 100)', '(4, 200)']

Parameters **mangled_list** (*List[Union[str,float]]*) – A list with strings representing tuples broken up by round brackets.

Returns A list containing the restores tuples (in string form) which can be converted back to numbered tuples using *list_string_to_tuple*

Return type List[str]

sanitize_yaml

`glotaran.builtin.io.yml.sanatize.sanitize_yaml(d: dict, do_keys: bool = True, do_values: bool = False) → dict`

Sanitize a yaml-returned dict for key or (list) values containing tuples

Parameters **d** (*dict*) – a dict resulting from parsing a pyglotaran model spec yml file

Returns a sanitized dict with (broken) string tuples restored as proper tuples

Return type dict

string_to_tuple

glotaran.builtin.io.yml.sanatize.**string_to_tuple**(*tuple_str: str, from_list=False*) → Union[Tuple[float], Tuple[str], float, str]

[summary]

Parameters

- **tuple_str** (*str*) – A string representing some tuple to convert the numbers inside the string tuple are mapped to float
- **from_list** (*bool, optional*) – only if true will a single number string be converted to float, otherwise returned as-is since it may represent a label, by default False

Returns Returns the tuple intended by the string

Return type Union[Tuple[float], Tuple[str], float, str]

yml

Classes

Summary

YmlProjectIo	Initialize a Project IO plugin with the name of the format.

YmlProjectIo

class glotaran.builtin.io.yml.yml.**YmlProjectIo**(*format_name: str*)

Bases: *glotaran.io.interface.ProjectIoInterface*

Initialize a Project IO plugin with the name of the format.

Parameters **format_name** (*str*) – Name of the supported format an instance uses.

Methods Summary

load_model	parse_yaml_file reads the given file and parses its content as YML.
load_parameters	Create a ParameterGroup instance from the specs defined in a file (**NOT IMPLEMENTED**).
load_result	Create a Result instance from the specs defined in a file (**NOT IMPLEMENTED**).
load_scheme	Create a Scheme instance from the specs defined in a file (**NOT IMPLEMENTED**).
save_model	Save a Model instance to a spec file (**NOT IMPLEMENTED**).

continues on next page

Table 53 – continued from previous page

save_parameters	Save a ParameterGroup instance to a spec file (**NOT IMPLEMENTED**).
save_result	Save a Result instance to a spec file (**NOT IMPLEMENTED**).
save_scheme	Save a Scheme instance to a spec file (**NOT IMPLEMENTED**).

load_model

YmlProjectIo.**load_model**(*file_name: str*) → Model

parse_yaml_file reads the given file and parses its content as YML.

Parameters **filename** (*str*) – filename is the of the file to parse.

Returns The content of the file as dictionary.

Return type *Model*

load_parameters

YmlProjectIo.**load_parameters**(*file_name: str*) → *glotaran.parameter.parameter_group.ParameterGroup*

Create a ParameterGroup instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **file_name** (*str*) – File containing the parameter specs.

Returns ParameterGroup instance created from the file.

Return type *ParameterGroup*

load_result

YmlProjectIo.**load_result**(*result_path: str*) → Result

Create a Result instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **result_path** (*str*) – Path containing the result data.

Returns Result instance created from the file.

Return type *Result*

load_scheme

YmlProjectIo.**load_scheme**(*file_name: str*) → *glotaran.project.scheme.Scheme*

Create a Scheme instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **file_name** (*str*) – File containing the parameter specs.

Returns

- *Scheme* – Scheme instance created from the file.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

save_model

YmlProjectIo.**save_model**(*model: Model*, *file_name: str*)

Save a Model instance to a spec file (**NOT IMPLEMENTED**).

Parameters

- **model** (Model) – Model instance to save to specs file.
- **file_name** (*str*) – File to write the model specs to.

save_parameters

YmlProjectIo.**save_parameters**(*parameters: ParameterGroup*, *file_name: str*)

Save a ParameterGroup instance to a spec file (**NOT IMPLEMENTED**).

Parameters

- **parameters** (ParameterGroup) – ParameterGroup instance to save to specs file.
- **file_name** (*str*) – File to write the parameter specs to.

save_result

YmlProjectIo.**save_result**(*result: Result*, *result_path: str*)

Save a Result instance to a spec file (**NOT IMPLEMENTED**).

Parameters

- **result** (Result) – Result instance to save to specs file.
- **result_path** (*str*) – Path to write the result data to.

save_scheme

YmlProjectIo.**save_scheme**(*scheme:* glotaran.project.scheme.Scheme, *file_name: str*)

Save a Scheme instance to a spec file (**NOT IMPLEMENTED**).

Parameters

- **scheme** (Scheme) – Scheme instance to save to specs file.
- **file_name** (*str*) – File to write the scheme specs to.

Methods Documentation

load_model(*file_name: str*) → Model

parse_yaml_file reads the given file and parses its content as YML.

Parameters **filename** (*str*) – filename is the of the file to parse.

Returns The content of the file as dictionary.

Return type *Model*

load_parameters(*file_name: str*) → *glotaran.parameter.parameter_group.ParameterGroup*

Create a ParameterGroup instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **file_name** (*str*) – File containing the parameter specs.

Returns ParameterGroup instance created from the file.

Return type *ParameterGroup*

load_result(*result_path: str*) → Result

Create a Result instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **result_path** (*str*) – Path containing the result data.

Returns Result instance created from the file.

Return type *Result*

load_scheme(*file_name: str*) → *glotaran.project.scheme.Scheme*

Create a Scheme instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **file_name** (*str*) – File containing the parameter specs.

Returns

- *Scheme* – Scheme instance created from the file.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

save_model(*model: Model*, *file_name: str*)

Save a Model instance to a spec file (**NOT IMPLEMENTED**).

Parameters

- **model** (`Model`) – Model instance to save to specs file.
- **file_name** (*str*) – File to write the model specs to.

save_parameters(*parameters: ParameterGroup*, *file_name: str*)

Save a ParameterGroup instance to a spec file (**NOT IMPLEMENTED**).

Parameters

- **parameters** (`ParameterGroup`) – ParameterGroup instance to save to specs file.
- **file_name** (*str*) – File to write the parameter specs to.

save_result(*result: Result*, *result_path: str*)

Save a Result instance to a spec file (**NOT IMPLEMENTED**).

Parameters

- **result** (`Result`) – Result instance to save to specs file.
- **result_path** (*str*) – Path to write the result data to.

save_scheme(*scheme:* glotaran.project.scheme.Scheme, *file_name: str*)

Save a Scheme instance to a spec file (**NOT IMPLEMENTED**).

Parameters

- **scheme** (`Scheme`) – Scheme instance to save to specs file.
- **file_name** (*str*) – File to write the scheme specs to.

models

Glotaran Models Package

Modules

glotaran.builtin.models.kinetic_image
glotaran.builtin.models.kinetic_spectrum
glotaran.builtin.models.spectral

kinetic_image

Modules

glotaran.builtin.models.kinetic_image.initial_concentration	This package contains the initial concentration item.
glotaran.builtin.models.kinetic_image.irf	This package contains irf items.
glotaran.builtin.models.kinetic_image.k_matrix	K-Matrix
glotaran.builtin.models.kinetic_image.kinetic_baseline_megacomplex	This package contains the kinetic megacomplex item.
glotaran.builtin.models.kinetic_image.kinetic_decay_megacomplex	This package contains the kinetic megacomplex item.
glotaran.builtin.models.kinetic_image.kinetic_image_dataset_descriptor	Kinetic Image Dataset Descriptor
glotaran.builtin.models.kinetic_image.kinetic_image_model	
glotaran.builtin.models.kinetic_image.kinetic_image_result	

initial_concentration

This package contains the initial concentration item.

Classes

Summary

InitialConcentration	An initial concentration describes the population of the compartments at the beginning of an experiment.

InitialConcentration

class glotaran.builtin.models.kinetic_image.initial_concentration.**InitialConcentration**

Bases: `object`

An initial concentration describes the population of the compartments at the beginning of an experiment.

Attributes Summary

compartments	
exclude_from_normalize	
label	
parameters	

compartments

InitialConcentration.**compartments**

exclude_from_normalize

InitialConcentration.**exclude_from_normalize**

label

InitialConcentration.**label**

parameters

InitialConcentration.**parameters**

Methods Summary

fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	
mprint	
normalized	
validate	

fill

`InitialConcentration.fill(model: Model, parameters: ParameterGroup) → cls`

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

`classmethod InitialConcentration.from_dict(values: dict) → cls`

from_list

`classmethod InitialConcentration.from_list(values: list) → cls`

mprint

`InitialConcentration.mprint(parameters: ParameterGroup = None, initial_parameters: ParameterGroup = None) → str`

normalized

`InitialConcentration.normalized() → glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcentration`

validate

`InitialConcentration.validate(model: Model, parameters=None) → list[str]`

Methods Documentation

`property compartments: List[str]`

`property exclude_from_normalize: List[str]`

`fill(model: Model, parameters: ParameterGroup) → cls`

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod **from_dict**(*values: dict*) → cls

classmethod **from_list**(*values: list*) → cls

property **label**: str

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

normalized() → *glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcentration*

property **parameters**: List[glotaran.parameter.parameter.Parameter]

validate(*model: Model*, *parameters=None*) → list[str]

irf

This package contains irf items.

Classes

Summary

Irf	Represents an IRF.
IrfGaussian	
IrfMeasured	A measured IRF.
IrfMultiGaussian	Represents a gaussian IRF.

Irf

class glotaran.builtin.models.kinetic_image.irf.**Irf**

Bases: object

Represents an IRF.

Methods Summary

add_type
get_default_type

add_type

classmethod `Irf.`**`add_type`**(*type_name: str*, *attribute_type: type*)

get_default_type

classmethod `Irf.`**`get_default_type`**() → str

Methods Documentation

classmethod **`add_type`**(*type_name: str*, *attribute_type: type*)

classmethod **`get_default_type`**() → str

IrfGaussian

class `glotaran.builtin.models.kinetic_image.irf.`**`IrfGaussian`**

Bases: *`glotaran.builtin.models.kinetic_image.irf.IrfMultiGaussian`*

Attributes Summary

backsweep
backsweep_period
center
label
normalize
scale
shift
type
width

backsweep

IrfGaussian.**backsweep**

backsweep_period

IrfGaussian.**backsweep_period**

center

IrfGaussian.**center**

label

IrfGaussian.**label**

normalize

IrfGaussian.**normalize**

scale

IrfGaussian.**scale**

shift

IrfGaussian.**shift**

type

IrfGaussian.**type**

width

IrfGaussian.**width**

Methods Summary

calculate	
fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	
mprint	
parameter	
validate	

calculate

IrfGaussian.**calculate**(*index: int*, *global_axis: numpy.ndarray*, *model_axis: numpy.ndarray*) → numpy.ndarray

fill

IrfGaussian.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

classmethod IrfGaussian.**from_dict**(*values: dict*) → cls

from_list

classmethod IrfGaussian.**from_list**(*values: list*) → cls

mprint

IrfGaussian.**mprint**(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

parameter

IrfGaussian.**parameter**(*global_index: int*, *global_axis: numpy.ndarray*) → Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray, float, bool, float]

validate

IrfGaussian.**validate**(*model: Model*, *parameters=None*) → list[str]

Methods Documentation

property backsweep: bool

property backsweep_period: glotaran.parameter.parameter.Parameter

calculate(*index: int*, *global_axis: numpy.ndarray*, *model_axis: numpy.ndarray*) → numpy.ndarray

property center: glotaran.parameter.parameter.Parameter

fill(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

property label: str

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property normalize: bool

parameter(*global_index: int*, *global_axis: numpy.ndarray*) → Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray, float, bool, float]

property scale: List[glotaran.parameter.parameter.Parameter]

property shift: List[glotaran.parameter.parameter.Parameter]

property type: str

validate(*model: Model*, *parameters=None*) → list[str]

property width: glotaran.parameter.parameter.Parameter

IrfMeasured

class glotaran.builtin.models.kinetic_image.irf.**IrfMeasured**

Bases: `object`

A measured IRF. The data must be supplied by the dataset.

Attributes Summary

label	
type	

label

IrfMeasured.**label**

type

IrfMeasured.**type**

Methods Summary

fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	
mprint	
validate	

fill

IrfMeasured.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

classmethod IrfMeasured.**from_dict**(*values: dict*) → cls

from_list

classmethod IrfMeasured.**from_list**(*values: list*) → cls

mprint

IrfMeasured.**mprint**(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

validate

IrfMeasured.**validate**(*model: Model*, *parameters=None*) → list[str]

Methods Documentation

fill(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

property label: str

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property type: str

validate(*model: Model*, *parameters=None*) → list[str]

IrfMultiGaussian

class glotaran.builtin.models.kinetic_image.irf.**IrfMultiGaussian**

Bases: `object`

Represents a gaussian IRF.

One width and one center is a single gauss.

One center and multiple widths is a multiple gaussian.

Multiple center and multiple widths is Double-, Triple- , etc. Gaussian.

Parameters

- **label** – label of the irf
- **center** – one or more center of the irf as parameter indices
- **width** – one or more widths of the gaussian as parameter index
- **center_dispersion** – polynomial coefficients for the dispersion of the center as list of parameter indices. None for no dispersion.
- **width_dispersion** – polynomial coefficients for the dispersion of the width as parameter indices. None for no dispersion.

Attributes Summary

backsweep
backsweep_period
center
label
normalize
scale
shift
type
width

backsweep

`IrfMultiGaussian.backsweep`

backsweep_period

`IrfMultiGaussian.backsweep_period`

center

`IrfMultiGaussian.center`

label

`IrfMultiGaussian.label`

normalize

`IrfMultiGaussian.normalize`

scale

`IrfMultiGaussian.scale`

shift

`IrfMultiGaussian.shift`

type

`IrfMultiGaussian.type`

width

`IrfMultiGaussian.width`

Methods Summary

calculate	
fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	
mprint	
parameter	
validate	

calculate

IrfMultiGaussian.**calculate**(*index: int*, *global_axis: numpy.ndarray*, *model_axis: numpy.ndarray*) → numpy.ndarray

fill

IrfMultiGaussian.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

classmethod IrfMultiGaussian.**from_dict**(*values: dict*) → cls

from_list

classmethod IrfMultiGaussian.**from_list**(*values: list*) → cls

mprint

IrfMultiGaussian.**mprint**(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

parameter

IrfMultiGaussian.**parameter**(*global_index: int*, *global_axis: numpy.ndarray*) → Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray, float, bool, float]

validate

IrfMultiGaussian.**validate**(*model: Model*, *parameters=None*) → list[str]

Methods Documentation

property backsweep: bool

property backsweep_period: glotaran.parameter.parameter.Parameter

calculate(*index: int*, *global_axis: numpy.ndarray*, *model_axis: numpy.ndarray*) → numpy.ndarray

property center: List[glotaran.parameter.parameter.Parameter]

fill(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

property label: str

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property normalize: bool

parameter(*global_index: int*, *global_axis: numpy.ndarray*) → Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray, float, bool, float]

property scale: List[glotaran.parameter.parameter.Parameter]

property shift: List[glotaran.parameter.parameter.Parameter]

property type: str

validate(*model: Model*, *parameters=None*) → list[str]

property width: List[glotaran.parameter.parameter.Parameter]

k_matrix

K-Matrix

Classes

Summary

KMatrix	A K-Matrix represents a first order differental system.

KMatrix

class glotaran.builtin.models.kinetic_image.k_matrix.**KMatrix**

Bases: object

A K-Matrix represents a first order differental system.

Attributes Summary

label
matrix

label

KMatrix.**label**

matrix

KMatrix.**matrix**

Methods Summary

a_matrix	The resulting A matrix of the KMatrix.
a_matrix_as_markdown	Returns the A Matrix as markdown formatted table.
a_matrix_non_unibranch	The resulting A matrix of the KMatrix for a non-unibranched model.
a_matrix_unibranch	The resulting A matrix of the KMatrix for an unibranched model.
combine	Creates a combined matrix.
eigen	Returns the eigenvalues and eigenvectors of the k matrix.
empty	Creates an empty K-Matrix.
fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	
full	The full representation of the KMatrix as numpy array.
involved_compartments	A list of all compartments in the Matrix.
is_unibranched	Returns true in the KMatrix represents an unibranched model.
matrix_as_markdown	Returns the KMatrix as markdown formatted table.
mprint	
rates	The resulting rates of the matrix.
reduced	The reduced representation of the KMatrix as numpy array.
validate	

a_matrix

KMatrix.**a_matrix**(*initial_concentration: glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcentration*) → numpy.ndarray

The resulting A matrix of the KMatrix.

Parameters **initial_concentration** – The initial concentration.

a_matrix_as_markdown

KMatrix.**a_matrix_as_markdown**(*initial_concentration: glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcentration*) → *glotaran.utils.ipython.MarkdownStr*

Returns the A Matrix as markdown formatted table.

Parameters **initial_concentration** – The initial concentration.

a_matrix_non_unibranch

KMatrix.**a_matrix_non_unibranch**(*initial_concentration: glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcentration*) → numpy.ndarray

The resulting A matrix of the KMatrix for a non-unibranched model.

Parameters **initial_concentration** – The initial concentration.

a_matrix_unibranch

KMatrix.**a_matrix_unibranch**(*initial_concentration: glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcentration*) → numpy.ndarray

The resulting A matrix of the KMatrix for an unibranched model.

Parameters **initial_concentration** – The initial concentration.

combine

KMatrix.**combine**(*k_matrix: glotaran.builtin.models.kinetic_image.k_matrix.KMatrix*) → *glotaran.builtin.models.kinetic_image.k_matrix.KMatrix*

Creates a combined matrix.

When combining k-matrices km1 and km2 (km1.combine(km2)), entries in km1 will be overwritten by corresponding entries in km2.

Parameters **k_matrix** – KMatrix to combine with.

Returns The combined KMatrix.

Return type combined

eigen

`KMatrix.eigen`(*compartments: list[str]*) → tuple[np.ndarray, np.ndarray]

Returns the eigenvalues and eigenvectors of the k matrix.

Parameters **compartments** – The compartment order.

empty

classmethod `KMatrix.empty`(*label: str*, *compartments: list[str]*) → KMatrix

Creates an empty K-Matrix. Useful for combining.

Parameters

- **label** – Label of the K-Matrix
- **compartments** – A list of all compartments in the model.

fill

`KMatrix.fill`(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

classmethod `KMatrix.from_dict`(*values: dict*) → cls

from_list

classmethod `KMatrix.from_list`(*values: list*) → cls

full

`KMatrix.full`(*compartments: list[str]*) → np.ndarray

The full representation of the KMatrix as numpy array.

Parameters **compartments** – The compartment order.

involved_compartments

`KMatrix.`**`involved_compartments`**`()` → list[str]

A list of all compartments in the Matrix.

is_unibranched

`KMatrix.`**`is_unibranched`**`(`*initial_concentration: glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcentration*`)` → bool

Returns true in the KMatrix represents an unibranched model.

Parameters **`initial_concentration`** – The initial concentration.

matrix_as_markdown

`KMatrix.`**`matrix_as_markdown`**`(`*compartments: list[str] = None*, *fill_parameters: bool = False*`)` → MarkdownStr

Returns the KMatrix as markdown formatted table.

Parameters

- **`compartments`** – (default = None) An optional list defining the desired order of compartments.
- **`fill_parameters`** (*bool*) – (default = False) If true, the entries will be filled with the actual parameter values instead of labels.

mprint

`KMatrix.`**`mprint`**`(`*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*`)` → str

rates

`KMatrix.`**`rates`**`(`*initial_concentration: glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcentration*`)` → numpy.ndarray

The resulting rates of the matrix.

Parameters **`initial_concentration`** – The initial concentration.

reduced

`KMatrix.`**`reduced`**`(`*compartments: list[str]*`)` → np.ndarray

The reduced representation of the KMatrix as numpy array.

Parameters **`compartments`** – The compartment order.

validate

KMatrix.**validate**(*model: Model*, *parameters=None*) → list[str]

Methods Documentation

a_matrix(*initial_concentration: glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcentration*) → numpy.ndarray
The resulting A matrix of the KMatrix.
Parameters **initial_concentration** – The initial concentration.

a_matrix_as_markdown(*initial_concentration: glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcentration*) → *glotaran.utils.ipython.MarkdownStr*
Returns the A Matrix as markdown formatted table.
Parameters **initial_concentration** – The initial concentration.

a_matrix_non_unibranch(*initial_concentration: glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcentration*) → numpy.ndarray
The resulting A matrix of the KMatrix for a non-unibranched model.
Parameters **initial_concentration** – The initial concentration.

a_matrix_unibranch(*initial_concentration: glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcentration*) → numpy.ndarray
The resulting A matrix of the KMatrix for an unibranched model.
Parameters **initial_concentration** – The initial concentration.

combine(*k_matrix: glotaran.builtin.models.kinetic_image.k_matrix.KMatrix*) → *glotaran.builtin.models.kinetic_image.k_matrix.KMatrix*
Creates a combined matrix.

When combining k-matrices km1 and km2 (km1.combine(km2)), entries in km1 will be overwritten by corresponding entries in km2.
Parameters **k_matrix** – KMatrix to combine with.
Returns The combined KMatrix.
Return type combined

eigen(*compartments: list[str]*) → tuple[np.ndarray, np.ndarray]
Returns the eigenvalues and eigenvectors of the k matrix.
Parameters **compartments** – The compartment order.

classmethod empty(*label: str*, *compartments: list[str]*) → *KMatrix*
Creates an empty K-Matrix. Useful for combining.
Parameters
- **label** – Label of the K-Matrix
- **compartments** – A list of all compartments in the model.

fill(*model: Model*, *parameters: ParameterGroup*) → cls
Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
Parameters
- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

full(*compartments: list[str]*) → np.ndarray

The full representation of the KMatrix as numpy array.

Parameters compartments – The compartment order.

involved_compartments() → list[str]

A list of all compartments in the Matrix.

is_unibranched(*initial_concentration: glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcentration*) → bool

Returns true in the KMatrix represents an unibranched model.

Parameters initial_concentration – The initial concentration.

property label: str

property matrix: Dict[Tuple[str, str], glotaran.parameter.parameter.Parameter]

matrix_as_markdown(*compartments: list[str] = None*, *fill_parameters: bool = False*) → MarkdownStr

Returns the KMatrix as markdown formatted table.

Parameters

- **compartments** – (default = None) An optional list defining the desired order of compartments.
- **fill_parameters** (*bool*) – (default = False) If true, the entries will be filled with the actual parameter values instead of labels.

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

rates(*initial_concentration: glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcentration*) → numpy.ndarray

The resulting rates of the matrix.

Parameters initial_concentration – The initial concentration.

reduced(*compartments: list[str]*) → np.ndarray

The reduced representation of the KMatrix as numpy array.

Parameters compartments – The compartment order.

validate(*model: Model*, *parameters=None*) → list[str]

kinetic_baseline_megacomplex

This package contains the kinetic megacomplex item.

Classes

Summary

KineticBaselineMegacomplex

KineticBaselineMegacomplex

class glotaran.builtin.models.kinetic_image.kinetic_baseline_megacomplex.**KineticBaselineMegacomplex**
Bases: *glotaran.model.megacomplex.Megacomplex*

Attributes Summary

label
type

label

KineticBaselineMegacomplex.**label**

type

KineticBaselineMegacomplex.**type**

Methods Summary

calculate_matrix	
fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	

continues on next page

Table 72 – continued from previous page

mprint	
validate	

calculate_matrix

KineticBaselineMegacomplex.**calculate_matrix**(*model*, *dataset_descriptor: DatasetDescriptor*, *indices: dict[str, int]*, *axis: dict[str, np.ndarray]*, ***kwargs*)

fill

KineticBaselineMegacomplex.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

classmethod KineticBaselineMegacomplex.**from_dict**(*values: dict*) → cls

from_list

classmethod KineticBaselineMegacomplex.**from_list**(*values: list*) → cls

mprint

KineticBaselineMegacomplex.**mprint**(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

validate

KineticBaselineMegacomplex.**validate**(*model: Model*, *parameters=None*) → list[str]

Methods Documentation

calculate_matrix(*model*, *dataset_descriptor: DatasetDescriptor*, *indices: dict[str, int]*, *axis: dict[str, np.ndarray]*, ***kwargs*)

fill(*model: Model*, *parameters: ParameterGroup*) → cls
Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters
- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod **from_dict**(*values: dict*) → cls

classmethod **from_list**(*values: list*) → cls

property **label**: str

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property **type**: str

validate(*model: Model*, *parameters=None*) → list[str]

kinetic_decay_megacomplex

This package contains the kinetic megacomplex item.

Functions

Summary

calculate_kinetic_matrix_gaussian_irf	Calculates a kinetic matrix with a gaussian irf.
calculate_kinetic_matrix_no_irf	
kinetic_image_matrix_implementation	

calculate_kinetic_matrix_gaussian_irf

glotaran.builtin.models.kinetic_image.kinetic_decay_megacomplex.**calculate_kinetic_matrix_gaussian_i**

Calculates a kinetic matrix with a gaussian irf.

calculate_kinetic_matrix_no_irf

glotaran.builtin.models.kinetic_image.kinetic_decay_megacomplex.**calculate_kinetic_matrix_no_irf**(*ma*
rat
tim

kinetic_image_matrix_implementation

glotaran.builtin.models.kinetic_image.kinetic_decay_megacomplex.**kinetic_image_matrix_implementatio**

Classes

Summary

KineticDecayMegacomplex	A Megacomplex with one or more K-Matrices.

KineticDecayMegacomplex

class glotaran.builtin.models.kinetic_image.kinetic_decay_megacomplex.**KineticDecayMegacomplex**

Bases: *glotaran.model.megacomplex.Megacomplex*

A Megacomplex with one or more K-Matrices.

Attributes Summary

involved_compartments	
k_matrix	
label	
type	

involved_compartments

KineticDecayMegacomplex.**involved_compartments**

k_matrix

KineticDecayMegacomplex.**k_matrix**

label

KineticDecayMegacomplex.**label**

type

KineticDecayMegacomplex.**type**

Methods Summary

calculate_matrix	
fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	

continues on next page

Table 76 – continued from previous page

from_list
full_k_matrix
has_k_matrix
mprint
validate

calculate_matrix

`KineticDecayMegacomplex.`**`calculate_matrix`**(*model*, *dataset_descriptor: DatasetDescriptor*, *indices: dict[str, int]*, *axis: dict[str, np.ndarray]*, ***kwargs*)

fill

`KineticDecayMegacomplex.`**`fill`**(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **`model`** – A glotaran model.
- **`parameter`** (`ParameterGroup`) – The parameter group to fill from.

from_dict

`classmethod` `KineticDecayMegacomplex.`**`from_dict`**(*values: dict*) → cls

from_list

`classmethod` `KineticDecayMegacomplex.`**`from_list`**(*values: list*) → cls

full_k_matrix

`KineticDecayMegacomplex.`**`full_k_matrix`**(*model=None*)

has_k_matrix

`KineticDecayMegacomplex.has_k_matrix()` → bool

mprint

`KineticDecayMegacomplex.mprint`(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

validate

`KineticDecayMegacomplex.validate`(*model: Model*, *parameters=None*) → list[str]

Methods Documentation

calculate_matrix(*model*, *dataset_descriptor: DatasetDescriptor*, *indices: dict[str, int]*, *axis: dict[str, np.ndarray]*, ***kwargs*)

fill(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

full_k_matrix(*model=None*)

has_k_matrix() → bool

property involved_compartments

property k_matrix: List[str]

property label: str

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property type: str

validate(*model: Model*, *parameters=None*) → list[str]

kinetic_image_dataset_descriptor

Kinetic Image Dataset Descriptor

Classes

Summary

KineticImageDatasetDescriptor

KineticImageDatasetDescriptor

class glotaran.builtin.models.kinetic_image.kinetic_image_dataset_descriptor.**KineticImageDatasetDescriptor**

Bases: *glotaran.model.dataset_descriptor.DatasetDescriptor*

Attributes Summary

initial_concentration
irf
label
megacomplex
megacomplex_scale
scale

initial_concentration

KineticImageDatasetDescriptor.**initial_concentration**

irf

`KineticImageDatasetDescriptor.irf`

label

`KineticImageDatasetDescriptor.label`

megacomplex

`KineticImageDatasetDescriptor.megacomplex`

megacomplex_scale

`KineticImageDatasetDescriptor.megacomplex_scale`

scale

`KineticImageDatasetDescriptor.scale`

Methods Summary

fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	
iterate_megacomplexes	
mprint	
validate	

fill

`KineticImageDatasetDescriptor.fill`(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

classmethod KineticImageDatasetDescriptor.**from_dict**(*values: dict*) → cls

from_list

classmethod KineticImageDatasetDescriptor.**from_list**(*values: list*) → cls

iterate_megacomplexes

KineticImageDatasetDescriptor.**iterate_megacomplexes**() → Generator[tuple[Parameter | int, Megacomplex | str]]

mprint

KineticImageDatasetDescriptor.**mprint**(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

validate

KineticImageDatasetDescriptor.**validate**(*model: Model*, *parameters=None*) → list[str]

Methods Documentation

fill(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

property initial_concentration: str

property irf: str

iterate_megacomplexes() → Generator[tuple[Parameter | int, Megacomplex | str]]

property label: str

property megacomplex: List[str]

property megacomplex_scale: List[glotaran.parameter.parameter.Parameter]

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property scale: glotaran.parameter.parameter.Parameter

validate(*model: Model*, *parameters=None*) → list[str]

kinetic_image_model

Functions

Summary

index_dependent

index_dependent

glotaran.builtin.models.kinetic_image.kinetic_image_model.**index_dependent**(*model:* glotaran.builtin.models.kinetic_i
→ bool

Classes

Summary

KineticImageModel

KineticImageModel

class glotaran.builtin.models.kinetic_image.kinetic_image_model.**KineticImageModel**

Bases: *glotaran.model.base_model.Model*

Attributes Summary

additional_penalty_function	
constrain_matrix_function	
dataset	
global_dimension	
global_matrix	
has_additional_penalty_function	
has_matrix_constraints_function	
initial_concentration	
irf	
k_matrix	
megacomplex	
model_dimension	
model_type	The type of the model as human readable string.
retrieve_clp_function	
weights	

additional_penalty_function

KineticImageModel.**additional_penalty_function = None**

constrain_matrix_function

`KineticImageModel.constrain_matrix_function = None`

dataset

`KineticImageModel.dataset`

global_dimension

`KineticImageModel.global_dimension = 'pixel'`

global_matrix

`KineticImageModel.global_matrix = None`

has_additional_penalty_function

`KineticImageModel.has_additional_penalty_function = None`

has_matrix_constraints_function

`KineticImageModel.has_matrix_constraints_function = None`

initial_concentration

`KineticImageModel.initial_concentration`

irf

`KineticImageModel.irf`

k_matrix

`KineticImageModel.k_matrix`

megacomplex

`KineticImageModel.megacomplex`

model_dimension

`KineticImageModel.model_dimension = 'time'`

model_type

`KineticImageModel.model_type`

The type of the model as human readable string.

retrieve_clp_function

`KineticImageModel.retrieve_clp_function = None`

weights

`KineticImageModel.weights`

Methods Summary

add_weights	
finalize_data	
from_dict	Creates a model from a dictionary.
get_dataset	
get_initial_concentration	
get_irf	
get_k_matrix	
get_megacomplex	
grouped	
index_dependent	
markdown	Formats the model as Markdown string.
problem_list	Returns a list with all problems in the model and missing parameters if specified.
set_dataset	

continues on next page

Table 83 – continued from previous page

set_initial_concentration	
set_irf	
set_k_matrix	
set_megacomplex	
simulate	Simulates the model.
valid	Returns *True* if the number problems in the model is 0, else *False*
validate	Returns a string listing all problems in the model and missing parameters if specified.

add_weights

`KineticImageModel.`**`add_weights`**`(`*item:* glotaran.model.weight.Weight`)`

finalize_data

`KineticImageModel.`**`finalize_data`**`(`*problem: Problem*, *data: dict[str, xr.Dataset]*`)`

from_dict

`classmethod` `KineticImageModel.`**`from_dict`**`(`*model_dict_ref: dict*`)` → *glotaran.model.base_model.Model*

Creates a model from a dictionary.

Parameters **`model_dict`** – Dictionary containing the model.

get_dataset

`KineticImageModel.`**`get_dataset`**`(`*label*`)` → *glotaran.builtin.models.kinetic_image.kinetic_image_dataset_descriptor.KineticImag*

get_initial_concentration

`KineticImageModel.`**`get_initial_concentration`**`(`*label*`)` → *glotaran.builtin.models.kinetic_image.initial_concentration.Initial*

get_irf

`KineticImageModel.get_irf(label)` → *glotaran.builtin.models.kinetic_image.irf.Irf*

get_k_matrix

`KineticImageModel.get_k_matrix(label)` → *glotaran.builtin.models.kinetic_image.k_matrix.KMatrix*

get_megacomplex

`KineticImageModel.get_megacomplex(label)` → glotaran.model.decorator._set_megacomplexes.<locals>.MetaMegacomplex

grouped

`KineticImageModel.grouped()`

index_dependent

`KineticImageModel.index_dependent()` → bool

markdown

`KineticImageModel.markdown(`*parameters: Optional[*glotaran.parameter.parameter_group.ParameterGroup*] = None*, *initial_parameters: Optional[*glotaran.parameter.parameter_group.ParameterGroup*] = None*, *base_heading_level:* int *= 1*`)` → *glotaran.utils.ipython.MarkdownStr*

Formats the model as Markdown string.

Parameters will be included if specified.

Parameters

- **parameter** (ParameterGroup) – Parameter to include.
- **initial_parameters** (ParameterGroup) – Initial values for the parameters.
- **base_heading_level** (*int*) – Base heading level of the markdown sections.

  E.g.:
  - If it is 1 the string will start with '# Model'.
  - If it is 3 the string will start with '### Model'.

problem_list

KineticImageModel.**problem_list**(*parameters: ParameterGroup = None*) → list[str]

Returns a list with all problems in the model and missing parameters if specified.

Parameters **parameter** – The parameter to validate.

set_dataset

KineticImageModel.**set_dataset**(*label*, *item:* glotaran.builtin.models.kinetic_image.kinetic_image_dataset_descriptor.KineticImag

set_initial_concentration

KineticImageModel.**set_initial_concentration**(*label*, *item:* glotaran.builtin.models.kinetic_image.initial_concentration.Initial

set_irf

KineticImageModel.**set_irf**(*label*, *item:* glotaran.builtin.models.kinetic_image.irf.Irf)

set_k_matrix

KineticImageModel.**set_k_matrix**(*label*, *item:* glotaran.builtin.models.kinetic_image.k_matrix.KMatrix)

set_megacomplex

KineticImageModel.**set_megacomplex**(*label*, *item: glotaran.model.decorator._set_megacomplexes.<locals>.MetaMegacomplex*)

simulate

KineticImageModel.**simulate**(*dataset: str*, *parameters: ParameterGroup*, *axes: dict[str, np.ndarray] = None*, *clp: np.ndarray | xr.DataArray = None*, *noise: bool = False*, *noise_std_dev: float = 1.0*, *noise_seed: int = None*) → xr.Dataset

Simulates the model.

Parameters

- **dataset** – Label of the dataset to simulate.
- **parameter** – The parameters for the simulation.

- **axes** – A dictionary with axes for simulation.
- **clp** – Conditionally linear parameters. Used instead of *model.global_matrix* if provided.
- **noise** – If *True* noise is added to the simulated data.
- **noise_std_dev** – The standard deviation of the noise.
- **noise_seed** – Seed for the noise.

valid

KineticImageModel.**valid**(*parameters: Optional[*glotaran.parameter.parameter_group.ParameterGroup*] = None*) → bool

Returns *True* if the number problems in the model is 0, else *False*

Parameters parameter – The parameter to validate.

validate

KineticImageModel.**validate**(*parameters: Optional[*glotaran.parameter.parameter_group.ParameterGroup*] = None*) → str

Returns a string listing all problems in the model and missing parameters if specified.

Parameters parameter – The parameter to validate.

Methods Documentation

add_weights(*item:* glotaran.model.weight.Weight)

additional_penalty_function = None

constrain_matrix_function = None

property dataset: Dict[str, glotaran.builtin.models.kinetic_image.kinetic_image_dataset_descriptor.KineticImageDatasetDescriptor]

finalize_data(*problem: Problem*, *data: dict[str, xr.Dataset]*)

classmethod from_dict(*model_dict_ref: dict*) → *glotaran.model.base_model.Model*

Creates a model from a dictionary.

Parameters model_dict – Dictionary containing the model.

get_dataset(*label*) → *glotaran.builtin.models.kinetic_image.kinetic_image_dataset_descriptor.KineticImageDatasetDescriptor*

get_initial_concentration(*label*) → *glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcentration*

get_irf(*label*) → *glotaran.builtin.models.kinetic_image.irf.Irf*

get_k_matrix(*label*) → *glotaran.builtin.models.kinetic_image.k_matrix.KMatrix*

get_megacomplex(*label*) →
glotaran.model.decorator._set_megacomplexes.<locals>.MetaMegacomplex

global_dimension = 'pixel'

global_matrix = None

grouped()

has_additional_penalty_function = None

has_matrix_constraints_function = None

index_dependent() → bool

property initial_concentration: Dict[str, glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcentration]

property irf: Dict[str, glotaran.builtin.models.kinetic_image.irf.Irf]

property k_matrix: Dict[str, glotaran.builtin.models.kinetic_image.k_matrix.KMatrix]

markdown(*parameters: Optional[*glotaran.parameter.parameter_group.ParameterGroup*] = None*, *initial_parameters: Optional[*glotaran.parameter.parameter_group.ParameterGroup*] = None*, *base_heading_level: int = 1*) → *glotaran.utils.ipython.MarkdownStr*

Formats the model as Markdown string.

Parameters will be included if specified.

Parameters

- **parameter** (ParameterGroup) – Parameter to include.
- **initial_parameters** (ParameterGroup) – Initial values for the parameters.
- **base_heading_level** (*int*) – Base heading level of the markdown sections.

  E.g.:

  – If it is 1 the string will start with '# Model'.

  – If it is 3 the string will start with '### Model'.

property megacomplex: Dict[str, glotaran.model.decorator._set_megacomplexes.<locals>.MetaMegacomplex]

model_dimension = 'time'

property model_type: str

The type of the model as human readable string.

problem_list(*parameters: ParameterGroup = None*) → list[str]

Returns a list with all problems in the model and missing parameters if specified.

Parameters parameter – The parameter to validate.

retrieve_clp_function = None

set_dataset(*label*, *item: glotaran.builtin.models.kinetic_image.kinetic_image_dataset_descriptor.KineticImageDatasetDescriptor*)

set_initial_concentration(*label*, *item: glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcentration*)

set_irf(*label*, *item: glotaran.builtin.models.kinetic_image.irf.Irf*)

set_k_matrix(*label*, *item: glotaran.builtin.models.kinetic_image.k_matrix.KMatrix*)

set_megacomplex(*label*, *item: glotaran.model.decorator._set_megacomplexes.<locals>.MetaMegacomplex*)

simulate(*dataset: str*, *parameters: ParameterGroup*, *axes: dict[str, np.ndarray] = None*, *clp: np.ndarray | xr.DataArray = None*, *noise: bool = False*, *noise_std_dev: float = 1.0*, *noise_seed: int = None*) → xr.Dataset

Simulates the model.

Parameters

- **dataset** – Label of the dataset to simulate.
- **parameter** – The parameters for the simulation.
- **axes** – A dictionary with axes for simulation.
- **clp** – Conditionally linear parameters. Used instead of *model.global_matrix* if provided.
- **noise** – If *True* noise is added to the simulated data.
- **noise_std_dev** – The standard deviation of the noise.
- **noise_seed** – Seed for the noise.

valid(*parameters: Optional[glotaran.parameter.parameter_group.ParameterGroup] = None*) → bool

Returns *True* if the number problems in the model is 0, else *False*

Parameters **parameter** – The parameter to validate.

validate(*parameters: Optional[glotaran.parameter.parameter_group.ParameterGroup] = None*) → str

Returns a string listing all problems in the model and missing parameters if specified.

Parameters **parameter** – The parameter to validate.

property weights: Dict[str, glotaran.model.weight.Weight]

kinetic_image_result

Functions

Summary

finalize_kinetic_image_result
retrieve_decay_associated_data
retrieve_irf
retrieve_species_associated_data

finalize_kinetic_image_result

`glotaran.builtin.models.kinetic_image.kinetic_image_result.`**`finalize_kinetic_image_result`**(*model*, *problem: Problem*, *data: dict[str, xr.Dataset]*)

retrieve_decay_associated_data

`glotaran.builtin.models.kinetic_image.kinetic_image_result.`**`retrieve_decay_associated_data`**(*model*, *dataset*, *dataset_de*, *name*)

retrieve_irf

`glotaran.builtin.models.kinetic_image.kinetic_image_result.`**`retrieve_irf`**(*model*, *dataset*, *dataset_descriptor*, *name*)

retrieve_species_associated_data

glotaran.builtin.models.kinetic_image.kinetic_image_result.**retrieve_species_associated_data**(*model*, *dataset*, *dataset_name*)

kinetic_spectrum

Modules

glotaran.builtin.models.kinetic_spectrum.coherent_artifact_megacomplex	This package contains the kinetic megacomplex item.
glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_dataset_descriptor	
glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model	
glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_result	
glotaran.builtin.models.kinetic_spectrum.spectral_constraints	This package contains compartment constraint items.
glotaran.builtin.models.kinetic_spectrum.spectral_irf	
glotaran.builtin.models.kinetic_spectrum.spectral_matrix	Glotaran Spectral Matrix
glotaran.builtin.models.kinetic_spectrum.spectral_penalties	This package contains compartment constraint items.
glotaran.builtin.models.kinetic_spectrum.spectral_relations	Glotaran Spectral Relation
glotaran.builtin.models.kinetic_spectrum.spectral_shape	This package contains the spectral shape item.

coherent_artifact_megacomplex

This package contains the kinetic megacomplex item.

Classes

Summary

CoherentArtifactMegacomplex

CoherentArtifactMegacomplex

class glotaran.builtin.models.kinetic_spectrum.coherent_artifact_megacomplex.**CoherentArtifactMegacomplex**

Bases: *glotaran.model.megacomplex.Megacomplex*

Attributes Summary

label	
order	
type	
width	

label

CoherentArtifactMegacomplex.**label**

order

CoherentArtifactMegacomplex.**order**

type

CoherentArtifactMegacomplex.**type**

width

CoherentArtifactMegacomplex.**width**

Methods Summary

calculate_matrix	
compartments	
fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	

continues on next page

Table 88 – continued from previous page

from_list
mprint
validate

calculate_matrix

`CoherentArtifactMegacomplex.calculate_matrix`(*model*, *dataset_descriptor: DatasetDescriptor*, *indices: dict[str, int]*, *axis: dict[str, np.ndarray]*, ***kwargs*)

compartments

`CoherentArtifactMegacomplex.compartments`()

fill

`CoherentArtifactMegacomplex.fill`(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

classmethod `CoherentArtifactMegacomplex.from_dict`(*values: dict*) → cls

from_list

classmethod `CoherentArtifactMegacomplex.from_list`(*values: list*) → cls

mprint

`CoherentArtifactMegacomplex.mprint(parameters: ParameterGroup = None, initial_parameters: ParameterGroup = None) → str`

validate

`CoherentArtifactMegacomplex.validate(model: Model, parameters=None) → list[str]`

Methods Documentation

calculate_matrix(*model*, *dataset_descriptor: DatasetDescriptor*, *indices: dict[str, int]*, *axis: dict[str, np.ndarray]*, ***kwargs*)

compartments()

fill(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

property label: str

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property order: int

property type: str

validate(*model: Model*, *parameters=None*) → list[str]

property width: glotaran.parameter.parameter.Parameter

kinetic_spectrum_dataset_descriptor

Classes

Summary

KineticSpectrumDatasetDescriptor

KineticSpectrumDatasetDescriptor

class glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_dataset_descriptor.**KineticSpectrumDatasetDescriptor**

Bases: *glotaran.builtin.models.kinetic_image.kinetic_image_dataset_descriptor.KineticImageDatasetDescriptor*

Attributes Summary

initial_concentration
irf
label
megacomplex
megacomplex_scale
scale
shape

initial_concentration

KineticSpectrumDatasetDescriptor.**initial_concentration**

irf

`KineticSpectrumDatasetDescriptor.irf`

label

`KineticSpectrumDatasetDescriptor.label`

megacomplex

`KineticSpectrumDatasetDescriptor.megacomplex`

megacomplex_scale

`KineticSpectrumDatasetDescriptor.megacomplex_scale`

scale

`KineticSpectrumDatasetDescriptor.scale`

shape

`KineticSpectrumDatasetDescriptor.shape`

Methods Summary

fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	
iterate_megacomplexes	
mprint	
validate	

fill

`KineticSpectrumDatasetDescriptor.fill(model: Model, parameters: ParameterGroup) → cls`

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

`classmethod KineticSpectrumDatasetDescriptor.from_dict(values: dict) → cls`

from_list

`classmethod KineticSpectrumDatasetDescriptor.from_list(values: list) → cls`

iterate_megacomplexes

`KineticSpectrumDatasetDescriptor.iterate_megacomplexes() → Generator[tuple[Parameter | int, Megacomplex | str]]`

mprint

`KineticSpectrumDatasetDescriptor.mprint(parameters: ParameterGroup = None, initial_parameters: ParameterGroup = None) → str`

validate

`KineticSpectrumDatasetDescriptor.validate(model: Model, parameters=None) → list[str]`

Methods Documentation

fill(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

property initial_concentration: str

property irf: str

iterate_megacomplexes() → Generator[tuple[Parameter | int, Megacomplex | str]]

property label: str

property megacomplex: List[str]

property megacomplex_scale: List[glotaran.parameter.parameter.Parameter]

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property scale: glotaran.parameter.parameter.Parameter

property shape: Dict[str, str]

validate(*model: Model*, *parameters=None*) → list[str]

kinetic_spectrum_model

Functions

Summary

apply_kinetic_model_constraints
apply_spectral_penalties
grouped
has_kinetic_model_constraints
has_spectral_penalties

continues on next page

Table 92 – continued from previous page

index_dependent	
retrieve_spectral_clps	
spectral_matrix	Calculates the matrix.

apply_kinetic_model_constraints

glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.**apply_kinetic_model_constraints**(*mo Ki- net- ic- Spe tru Mo dat str, pa- ram e- ters Pa- ram e- ter- Gro clp list, ma- trix np. in- dex floa* → tu- ple np.

apply_spectral_penalties

`glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.apply_spectral_penalties`(*model: KineticSpectrumModel, parameters: ParameterGroup, clp_labels: dict[str, list[str] | list[list[str]], clps: dict[str, list[np.ndarr*, *matrices: dict[str, np.ndarray | list[np.ndarr*, *data: dict[str, xr.Dataset]*, *group_tolera*: *float*) → np.ndarray

grouped

glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.**grouped**(*model:* glotaran.builtin.models.kinetic_spe

has_kinetic_model_constraints

glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.**has_kinetic_model_constraints**(*model* glotar → bool

has_spectral_penalties

glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.**has_spectral_penalties**(*model: Ki-net-ic-Spec-trum-Model*) → bool

index_dependent

glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.**index_dependent**(*model:* glotaran.builtin.models.k → bool

retrieve_spectral_clps

glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.**retrieve_spectral_clps**(*model: KineticSpectrumModel, parameters: ParameterGroup, clp_labels: dict[str, list[str] | list[list[str]]], reduced_clp_labels: dict[str, list[str] | list[list[str]]], reduced_clps: dict[str, list[np.ndarray]], data: dict[str, xr.Dataset]*) → dict[str, list[np.ndarray]]

spectral_matrix

glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.**spectral_matrix**(*dataset, axis*)

Calculates the matrix.

Parameters

- **matrix** (*np.array*) – The preallocated matrix.
- **compartment_order** (*list(str)*) – A list of compartment labels to map compartments to indices in the matrix.
- **parameter** (*glotaran.model.ParameterGroup*) –

Classes

Summary

KineticSpectrumModel	

KineticSpectrumModel

class glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_model.**KineticSpectrumModel**

Bases: *glotaran.builtin.models.kinetic_image.kinetic_image_model.KineticImageModel*

Attributes Summary

dataset	
equal_area_penalties	
global_dimension	
initial_concentration	
irf	
k_matrix	
megacomplex	
model_dimension	
model_type	The type of the model as human readable string.
shape	
spectral_constraints	
spectral_relations	
weights	

dataset

KineticSpectrumModel.**dataset**

equal_area_penalties

KineticSpectrumModel.**equal_area_penalties**

global_dimension

KineticSpectrumModel.**global_dimension = 'spectral'**

initial_concentration

KineticSpectrumModel.**initial_concentration**

irf

KineticSpectrumModel.**irf**

k_matrix

KineticSpectrumModel.**k_matrix**

megacomplex

KineticSpectrumModel.**megacomplex**

model_dimension

KineticSpectrumModel.**model_dimension = 'time'**

model_type

KineticSpectrumModel.**model_type**

The type of the model as human readable string.

shape

KineticSpectrumModel.**shape**

spectral_constraints

KineticSpectrumModel.**spectral_constraints**

spectral_relations

KineticSpectrumModel.**spectral_relations**

weights

KineticSpectrumModel.**weights**

Methods Summary

add_equal_area_penalties	
add_spectral_constraints	
add_spectral_relations	
add_weights	
additional_penalty_function	
constrain_matrix_function	
finalize_data	
from_dict	Creates a model from a dictionary.
get_dataset	
get_initial_concentration	
get_irf	
get_k_matrix	
get_megacomplex	
get_shape	
global_matrix	Calculates the matrix.
grouped	

continues on next page

Table 95 – continued from previous page

has_additional_penalty_function	
has_matrix_constraints_function	
index_dependent	
markdown	Formats the model as Markdown string.
problem_list	Returns a list with all problems in the model and missing parameters if specified.
retrieve_clp_function	
set_dataset	
set_initial_concentration	
set_irf	
set_k_matrix	
set_megacomplex	
set_shape	
simulate	Simulates the model.
valid	Returns *True* if the number problems in the model is 0, else *False*
validate	Returns a string listing all problems in the model and missing parameters if specified.

add_equal_area_penalties

KineticSpectrumModel.**add_equal_area_penalties**(*item:*
glotaran.builtin.models.kinetic_spectrum.spectral_penalties.Eq

add_spectral_constraints

KineticSpectrumModel.**add_spectral_constraints**(*item:*
glotaran.builtin.models.kinetic_spectrum.spectral_constraints.S

add_spectral_relations

KineticSpectrumModel.**add_spectral_relations**(*item: glotaran.builtin.models.kinetic_spectrum.spectral_relations.Spect*

add_weights

KineticSpectrumModel.**add_weights**(*item:* glotaran.model.weight.Weight)

additional_penalty_function

KineticSpectrumModel.**additional_penalty_function**(*parameters: ParameterGroup, clp_labels: dict[str, list[str] | list[list[str]]], clps: dict[str, list[np.ndarray]], matrices: dict[str, np.ndarray | list[np.ndarray]], data: dict[str, xr.Dataset], group_tolerance: float*) → np.ndarray

constrain_matrix_function

KineticSpectrumModel.**constrain_matrix_function**(*dataset: str, parameters: ParameterGroup, clp_labels: list[str], matrix: np.ndarray, index: float*) → tuple[list[str], np.ndarray]

finalize_data

KineticSpectrumModel.**finalize_data**(*problem: Problem, data: dict[str, xr.Dataset]*)

from_dict

classmethod KineticSpectrumModel.**from_dict**(*model_dict_ref: dict*) → *glotaran.model.base_model.Model*

Creates a model from a dictionary.

Parameters **model_dict** – Dictionary containing the model.

get_dataset

KineticSpectrumModel.**get_dataset**(*label*) → *glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_dataset_descriptor.I*

get_initial_concentration

KineticSpectrumModel.**get_initial_concentration**(*label*) → *glotaran.builtin.models.kinetic_image.initial_concentration.I*

get_irf

KineticSpectrumModel.**get_irf**(*label*) → *glotaran.builtin.models.kinetic_image.irf.Irf*

get_k_matrix

KineticSpectrumModel.**get_k_matrix**(*label*) → *glotaran.builtin.models.kinetic_image.k_matrix.KMatrix*

get_megacomplex

KineticSpectrumModel.**get_megacomplex**(*label*) → glotaran.model.decorator._set_megacomplexes.<locals>.MetaMegacomple

get_shape

KineticSpectrumModel.**get_shape**(*label*) → *glotaran.builtin.models.kinetic_spectrum.spectral_shape.SpectralShape*

global_matrix

static KineticSpectrumModel.**global_matrix**(*dataset*, *axis*)

Calculates the matrix.

Parameters

- **matrix** (*np.array*) – The preallocated matrix.
- **compartment_order** (*list(str)*) – A list of compartment labels to map compartments to indices in the matrix.
- **parameter** (*glotaran.model.ParameterGroup*) –

grouped

`KineticSpectrumModel.grouped()`

has_additional_penalty_function

`KineticSpectrumModel.has_additional_penalty_function()` → bool

has_matrix_constraints_function

`KineticSpectrumModel.has_matrix_constraints_function()` → bool

index_dependent

`KineticSpectrumModel.index_dependent()` → bool

markdown

`KineticSpectrumModel.markdown`(*parameters: Optional[*glotaran.parameter.parameter_group.ParameterGroup*] = None*, *initial_parameters: Optional[*glotaran.parameter.parameter_group.ParameterGroup*] = None*, *base_heading_level: int = 1*) → *glotaran.utils.ipython.MarkdownStr*

Formats the model as Markdown string.

Parameters will be included if specified.

Parameters

- **parameter** (ParameterGroup) – Parameter to include.
- **initial_parameters** (ParameterGroup) – Initial values for the parameters.
- **base_heading_level** (*int*) – Base heading level of the markdown sections.

  E.g.:

  – If it is 1 the string will start with '# Model'.

  – If it is 3 the string will start with '### Model'.

problem_list

`KineticSpectrumModel.`**`problem_list`**(*parameters: ParameterGroup = None*) → list[str]

Returns a list with all problems in the model and missing parameters if specified.

Parameters **`parameter`** – The parameter to validate.

retrieve_clp_function

`KineticSpectrumModel.`**`retrieve_clp_function`**(*parameters: ParameterGroup, clp_labels: dict[str, list[str] | list[list[str]]], reduced_clp_labels: dict[str, list[str] | list[list[str]]], reduced_clps: dict[str, list[np.ndarray]], data: dict[str, xr.Dataset]*) → dict[str, list[np.ndarray]]

set_dataset

`KineticSpectrumModel.`**`set_dataset`**(*label, item:* glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_dataset_descriptor.K

set_initial_concentration

`KineticSpectrumModel.`**`set_initial_concentration`**(*label, item:* glotaran.builtin.models.kinetic_image.initial_concentration.In

set_irf

`KineticSpectrumModel.`**`set_irf`**(*label, item:* glotaran.builtin.models.kinetic_image.irf.Irf)

set_k_matrix

`KineticSpectrumModel.`**`set_k_matrix`**(*label, item:* glotaran.builtin.models.kinetic_image.k_matrix.KMatrix)

set_megacomplex

KineticSpectrumModel.**set_megacomplex**(*label*, *item: glotaran.model.decorator._set_megacomplexes.<locals>.MetaMegacompl*

set_shape

KineticSpectrumModel.**set_shape**(*label*, *item:* glotaran.builtin.models.kinetic_spectrum.spectral_shape.SpectralShape)

simulate

KineticSpectrumModel.**simulate**(*dataset: str*, *parameters: ParameterGroup*, *axes: dict[str, np.ndarray] = None*, *clp: np.ndarray | xr.DataArray = None*, *noise: bool = False*, *noise_std_dev: float = 1.0*, *noise_seed: int = None*) → xr.Dataset

Simulates the model.

Parameters

- **dataset** – Label of the dataset to simulate.
- **parameter** – The parameters for the simulation.
- **axes** – A dictionary with axes for simulation.
- **clp** – Conditionally linear parameters. Used instead of *model.global_matrix* if provided.
- **noise** – If *True* noise is added to the simulated data.
- **noise_std_dev** – The standard deviation of the noise.
- **noise_seed** – Seed for the noise.

valid

KineticSpectrumModel.**valid**(*parameters: Optional[*glotaran.parameter.parameter_group.ParameterGroup*] = None*) → bool

Returns *True* if the number problems in the model is 0, else *False*

Parameters **parameter** – The parameter to validate.

validate

KineticSpectrumModel.**validate**(*parameters: Optional[*glotaran.parameter.parameter_group.ParameterGroup*] = None*) → str

Returns a string listing all problems in the model and missing parameters if specified.

Parameters **parameter** – The parameter to validate.

Methods Documentation

add_equal_area_penalties(*item:* glotaran.builtin.models.kinetic_spectrum.spectral_penalties.EqualAreaPenalty)

add_spectral_constraints(*item:* glotaran.builtin.models.kinetic_spectrum.spectral_constraints.SpectralConstraint)

add_spectral_relations(*item:* glotaran.builtin.models.kinetic_spectrum.spectral_relations.SpectralRelation)

add_weights(*item:* glotaran.model.weight.Weight)

additional_penalty_function(*parameters: ParameterGroup*, *clp_labels: dict[str, list[str] | list[list[str]]]*, *clps: dict[str, list[np.ndarray]]*, *matrices: dict[str, np.ndarray | list[np.ndarray]]*, *data: dict[str, xr.Dataset]*, *group_tolerance: float*) → np.ndarray

constrain_matrix_function(*dataset: str*, *parameters: ParameterGroup*, *clp_labels: list[str]*, *matrix: np.ndarray*, *index: float*) → tuple[list[str], np.ndarray]

property dataset: Dict[str, glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_dataset_descriptor.KineticSpectrumDatasetDescriptor]

property equal_area_penalties: Dict[str, glotaran.builtin.models.kinetic_spectrum.spectral_penalties.EqualAreaPenalty]

finalize_data(*problem: Problem*, *data: dict[str, xr.Dataset]*)

classmethod from_dict(*model_dict_ref: dict*) → *glotaran.model.base_model.Model*

Creates a model from a dictionary.

Parameters **model_dict** – Dictionary containing the model.

get_dataset(*label*) → *glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_dataset_descriptor.KineticSpectrumDatasetDes*

get_initial_concentration(*label*) → *glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcentration*

get_irf(*label*) → *glotaran.builtin.models.kinetic_image.irf.Irf*

get_k_matrix(*label*) → *glotaran.builtin.models.kinetic_image.k_matrix.KMatrix*

get_megacomplex(*label*) →
glotaran.model.decorator._set_megacomplexes.<locals>.MetaMegacomplex

get_shape(*label*) → *glotaran.builtin.models.kinetic_spectrum.spectral_shape.SpectralShape*

global_dimension = 'spectral'

static **global_matrix**(*dataset*, *axis*)
Calculates the matrix.

Parameters

- **matrix** (*np.array*) – The preallocated matrix.
- **compartment_order** (*list(str)*) – A list of compartment labels to map compartments to indices in the matrix.
- **parameter** (*glotaran.model.ParameterGroup*) –

grouped()

has_additional_penalty_function() → bool

has_matrix_constraints_function() → bool

index_dependent() → bool

property **initial_concentration: Dict[str, glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcentration]**

property **irf: Dict[str, glotaran.builtin.models.kinetic_image.irf.Irf]**

property **k_matrix: Dict[str, glotaran.builtin.models.kinetic_image.k_matrix.KMatrix]**

markdown(*parameters: Optional[*glotaran.parameter.parameter_group.ParameterGroup*] = None*, *initial_parameters: Optional[*glotaran.parameter.parameter_group.ParameterGroup*] = None*, *base_heading_level: int = 1*) → *glotaran.utils.ipython.MarkdownStr*
Formats the model as Markdown string.

Parameters will be included if specified.

Parameters

- **parameter** (ParameterGroup) – Parameter to include.
- **initial_parameters** (ParameterGroup) – Initial values for the parameters.
- **base_heading_level** (*int*) – Base heading level of the markdown sections.

  E.g.:

  – If it is 1 the string will start with '# Model'.

  – If it is 3 the string will start with '### Model'.

property megacomplex: Dict[str, glotaran.model.decorator._set_megacomplexes.<locals>.MetaMegacomplex]

model_dimension = 'time'

property model_type: str

The type of the model as human readable string.

problem_list(*parameters: ParameterGroup = None*) → list[str]

Returns a list with all problems in the model and missing parameters if specified.

Parameters **parameter** – The parameter to validate.

retrieve_clp_function(*parameters: ParameterGroup, clp_labels: dict[str, list[str] | list[list[str]]], reduced_clp_labels: dict[str, list[str] | list[list[str]]], reduced_clps: dict[str, list[np.ndarray]], data: dict[str, xr.Dataset]*) → dict[str, list[np.ndarray]]

set_dataset(*label, item: glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_dataset_descriptor.KineticSpectrumDatasetDes*

set_initial_concentration(*label, item: glotaran.builtin.models.kinetic_image.initial_concentration.InitialConcentration*)

set_irf(*label, item: glotaran.builtin.models.kinetic_image.irf.Irf*)

set_k_matrix(*label, item: glotaran.builtin.models.kinetic_image.k_matrix.KMatrix*)

set_megacomplex(*label, item: glotaran.model.decorator._set_megacomplexes.<locals>.MetaMegacomplex*)

set_shape(*label, item: glotaran.builtin.models.kinetic_spectrum.spectral_shape.SpectralShape*)

property shape: Dict[str, glotaran.builtin.models.kinetic_spectrum.spectral_shape.SpectralShape]

simulate(*dataset: str, parameters: ParameterGroup, axes: dict[str, np.ndarray] = None, clp: np.ndarray | xr.DataArray = None, noise: bool = False, noise_std_dev: float = 1.0, noise_seed: int = None*) → xr.Dataset

Simulates the model.

Parameters

- **dataset** – Label of the dataset to simulate.
- **parameter** – The parameters for the simulation.
- **axes** – A dictionary with axes for simulation.
- **clp** – Conditionally linear parameters. Used instead of *model.global_matrix* if provided.
- **noise** – If *True* noise is added to the simulated data.
- **noise_std_dev** – The standard deviation of the noise.
- **noise_seed** – Seed for the noise.

property spectral_constraints: Dict[str, glotaran.builtin.models.kinetic_spectrum.spectral_constraints.SpectralConstraint]

property spectral_relations: Dict[str, glotaran.builtin.models.kinetic_spectrum.spectral_relations.SpectralRelation]

valid(*parameters: Optional[*glotaran.parameter.parameter_group.ParameterGroup*] = None*) → bool

Returns *True* if the number problems in the model is 0, else *False*

Parameters **parameter** – The parameter to validate.

validate(*parameters: Optional[*glotaran.parameter.parameter_group.ParameterGroup*] = None*) → str

Returns a string listing all problems in the model and missing parameters if specified.

Parameters **parameter** – The parameter to validate.

property weights: Dict[str, glotaran.model.weight.Weight]

kinetic_spectrum_result

Functions

Summary

finalize_kinetic_spectrum_result

finalize_kinetic_spectrum_result

glotaran.builtin.models.kinetic_spectrum.kinetic_spectrum_result.**finalize_kinetic_spectrum_result**(*r*

spectral_constraints

This package contains compartment constraint items.

Functions

Summary

apply_spectral_constraints	

apply_spectral_constraints

`glotaran.builtin.models.kinetic_spectrum.spectral_constraints.`**`apply_spectral_constraints`**(*model: KineticSpectrumModel*, *clp_labels: list[str]*, *matrix: np.ndarray*, *index: float*) → tuple[list[str], np.ndarray]

Classes

Summary

OnlyConstraint	A only constraint sets the calculated matrix row of a compartment to 0 outside the given intervals.
SpectralConstraint	A compartment constraint is applied on one compartment on one or many intervals on the estimated axis type.
ZeroConstraint	A zero constraint sets the calculated matrix row of a compartment to 0 in the given intervals.

OnlyConstraint

class
glotaran.builtin.models.kinetic_spectrum.spectral_constraints.**OnlyConstraint**

Bases: `object`

A only constraint sets the calculated matrix row of a compartment to 0 outside the given intervals.

Attributes Summary

compartment	
interval	
type	

compartment

OnlyConstraint.**compartment**

interval

OnlyConstraint.**interval**

type

OnlyConstraint.**type**

Methods Summary

applies	Returns true if the index is in one of the intervals.
fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	
mprint	
validate	

applies

`OnlyConstraint.`**`applies`**`(`*index: Any*`)` → bool

Returns true if the index is in one of the intervals.

Parameters **`index`** –

Returns **applies**

Return type bool

fill

`OnlyConstraint.`**`fill`**`(`*model: Model*, *parameters: ParameterGroup*`)` → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **`model`** – A glotaran model.
- **`parameter`** (`ParameterGroup`) – The parameter group to fill from.

from_dict

`classmethod` `OnlyConstraint.`**`from_dict`**`(`*values: dict*`)` → cls

from_list

`classmethod` `OnlyConstraint.`**`from_list`**`(`*values: list*`)` → cls

mprint

`OnlyConstraint.`**`mprint`**`(`*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*`)` → str

validate

`OnlyConstraint.`**`validate`**`(`*model: Model*, *parameters=None*`)` → list[str]

Methods Documentation

applies(*index: Any*) → bool

Returns true if the index is in one of the intervals.

Parameters **index** –

Returns applies

Return type bool

property compartment: str

fill(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

property interval: List[Tuple[float, float]]

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property type: str

validate(*model: Model*, *parameters=None*) → list[str]

SpectralConstraint

class glotaran.builtin.models.kinetic_spectrum.spectral_constraints.**SpectralConstraint**

Bases: object

A compartment constraint is applied on one compartment on one or many intervals on the estimated axis type.

There are three types: zero, equal and equal area. See the documentation of the respective classes for details.

Methods Summary

add_type	
get_default_type	

add_type

classmethod SpectralConstraint.**add_type**(*type_name: str*, *attribute_type: type*)

get_default_type

classmethod SpectralConstraint.**get_default_type**() → str

Methods Documentation

classmethod add_type(*type_name: str*, *attribute_type: type*)

classmethod get_default_type() → str

ZeroConstraint

class
glotaran.builtin.models.kinetic_spectrum.spectral_constraints.**ZeroConstraint**

Bases: `object`

A zero constraint sets the calculated matrix row of a compartment to 0 in the given intervals.

Attributes Summary

compartment	
interval	
type	

compartment

ZeroConstraint.**compartment**

interval

ZeroConstraint.**interval**

type

ZeroConstraint.**type**

Methods Summary

applies	Returns true if the indexx is in one of the intervals.
fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	
mprint	
validate	

applies

ZeroConstraint.**applies**(*index: Any*) → bool

Returns true if the indexx is in one of the intervals.

Parameters **index** –

Returns applies

Return type bool

fill

`ZeroConstraint.fill(model: Model, parameters: ParameterGroup) → cls`

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

`classmethod ZeroConstraint.from_dict(values: dict) → cls`

from_list

`classmethod ZeroConstraint.from_list(values: list) → cls`

mprint

`ZeroConstraint.mprint(parameters: ParameterGroup = None, initial_parameters: ParameterGroup = None) → str`

validate

`ZeroConstraint.validate(model: Model, parameters=None) → list[str]`

Methods Documentation

`applies(index: Any) → bool`

Returns true if the indexx is in one of the intervals.

Parameters **index** –

Returns **applies**

Return type bool

`property compartment: str`

`fill(model: Model, parameters: ParameterGroup) → cls`

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.

- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

property interval: List[Tuple[float, float]]

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property type: str

validate(*model: Model*, *parameters=None*) → list[str]

spectral_irf

Classes

Summary

IrfSpectralGaussian	
IrfSpectralMultiGaussian	Represents a gaussian IRF.

IrfSpectralGaussian

class glotaran.builtin.models.kinetic_spectrum.spectral_irf.**IrfSpectralGaussian**

Bases: *glotaran.builtin.models.kinetic_spectrum.spectral_irf.IrfSpectralMultiGaussian*

Attributes Summary

backsweep
backsweep_period
center
center_dispersion
dispersion_center
label
model_dispersion_with_wavenumber

continues on next page

Table 105 – continued from previous page

normalize
scale
shift
type
width
width_dispersion

backsweep

IrfSpectralGaussian.**backsweep**

backsweep_period

IrfSpectralGaussian.**backsweep_period**

center

IrfSpectralGaussian.**center**

center_dispersion

IrfSpectralGaussian.**center_dispersion**

dispersion_center

IrfSpectralGaussian.**dispersion_center**

label

IrfSpectralGaussian.**label**

model_dispersion_with_wavenumber

IrfSpectralGaussian.**model_dispersion_with_wavenumber**

normalize

IrfSpectralGaussian.**normalize**

scale

IrfSpectralGaussian.**scale**

shift

IrfSpectralGaussian.**shift**

type

IrfSpectralGaussian.**type**

width

IrfSpectralGaussian.**width**

width_dispersion

IrfSpectralGaussian.**width_dispersion**

Methods Summary

calculate	
calculate_dispersion	
fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	
mprint	

continues on next page

Table 106 – continued from previous page

parameter	
validate	

calculate

`IrfSpectralGaussian.`**`calculate`**`(`*index: int*, *global_axis: numpy.ndarray*, *model_axis: numpy.ndarray*`)` → numpy.ndarray

calculate_dispersion

`IrfSpectralGaussian.`**`calculate_dispersion`**`(`*axis*`)`

fill

`IrfSpectralGaussian.`**`fill`**`(`*model: Model*, *parameters: ParameterGroup*`)` → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **`model`** – A glotaran model.
- **`parameter`** (`ParameterGroup`) – The parameter group to fill from.

from_dict

`classmethod` `IrfSpectralGaussian.`**`from_dict`**`(`*values: dict*`)` → cls

from_list

`classmethod` `IrfSpectralGaussian.`**`from_list`**`(`*values: list*`)` → cls

mprint

`IrfSpectralGaussian.`**`mprint`**`(`*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*`)` → str

parameter

IrfSpectralGaussian.**parameter**(*global_index: int*, *global_axis: numpy.ndarray*)

validate

IrfSpectralGaussian.**validate**(*model: Model*, *parameters=None*) → list[str]

Methods Documentation

property backsweep: bool

property backsweep_period: glotaran.parameter.parameter.Parameter

calculate(*index: int*, *global_axis: numpy.ndarray*, *model_axis: numpy.ndarray*) → numpy.ndarray

calculate_dispersion(*axis*)

property center: glotaran.parameter.parameter.Parameter

property center_dispersion: List[glotaran.parameter.parameter.Parameter]

property dispersion_center: glotaran.parameter.parameter.Parameter

fill(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

property label: str

property model_dispersion_with_wavenumber: bool

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property normalize: bool

parameter(*global_index: int*, *global_axis: numpy.ndarray*)

property scale: List[glotaran.parameter.parameter.Parameter]

property shift: List[glotaran.parameter.parameter.Parameter]

property type: str

validate(*model: Model*, *parameters=None*) → list[str]

property width: glotaran.parameter.parameter.Parameter

property width_dispersion: List[glotaran.parameter.parameter.Parameter]

IrfSpectralMultiGaussian

class
glotaran.builtin.models.kinetic_spectrum.spectral_irf.**IrfSpectralMultiGaussian**

Bases: *glotaran.builtin.models.kinetic_image.irf.IrfMultiGaussian*

Represents a gaussian IRF.

One width and one center is a single gauss.

One center and multiple widths is a multiple gaussian.

Multiple center and multiple widths is Double-, Triple- , etc. Gaussian.

Parameters

- **label** – label of the irf
- **center** – one or more center of the irf as parameter indices
- **width** – one or more widths of the gaussian as parameter index
- **center_dispersion** – polynomial coefficients for the dispersion of the center as list of parameter indices. None for no dispersion.
- **width_dispersion** – polynomial coefficients for the dispersion of the width as parameter indices. None for no dispersion.

Attributes Summary

backsweep
backsweep_period
center
center_dispersion
dispersion_center
label
model_dispersion_with_wavenumber
normalize
scale

continues on next page

Table 107 – continued from previous page

shift
type
width
width_dispersion

backsweep

IrfSpectralMultiGaussian.**backsweep**

backsweep_period

IrfSpectralMultiGaussian.**backsweep_period**

center

IrfSpectralMultiGaussian.**center**

center_dispersion

IrfSpectralMultiGaussian.**center_dispersion**

dispersion_center

IrfSpectralMultiGaussian.**dispersion_center**

label

IrfSpectralMultiGaussian.**label**

model_dispersion_with_wavenumber

IrfSpectralMultiGaussian.**model_dispersion_with_wavenumber**

normalize

IrfSpectralMultiGaussian.**normalize**

scale

IrfSpectralMultiGaussian.**scale**

shift

IrfSpectralMultiGaussian.**shift**

type

IrfSpectralMultiGaussian.**type**

width

IrfSpectralMultiGaussian.**width**

width_dispersion

IrfSpectralMultiGaussian.**width_dispersion**

Methods Summary

calculate	
calculate_dispersion	
fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	
mprint	
parameter	
validate	

calculate

IrfSpectralMultiGaussian.**calculate**(*index: int*, *global_axis: numpy.ndarray*, *model_axis: numpy.ndarray*) → numpy.ndarray

calculate_dispersion

IrfSpectralMultiGaussian.**calculate_dispersion**(*axis*)

fill

IrfSpectralMultiGaussian.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

classmethod IrfSpectralMultiGaussian.**from_dict**(*values: dict*) → cls

from_list

classmethod IrfSpectralMultiGaussian.**from_list**(*values: list*) → cls

mprint

IrfSpectralMultiGaussian.**mprint**(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

parameter

IrfSpectralMultiGaussian.**parameter**(*global_index: int*, *global_axis: numpy.ndarray*)

validate

IrfSpectralMultiGaussian.**validate**(*model: Model*, *parameters=None*) → list[str]

Methods Documentation

property backsweep: bool

property backsweep_period: glotaran.parameter.parameter.Parameter

calculate(*index: int*, *global_axis: numpy.ndarray*, *model_axis: numpy.ndarray*) → numpy.ndarray

calculate_dispersion(*axis*)

property center: List[glotaran.parameter.parameter.Parameter]

property center_dispersion: List[glotaran.parameter.parameter.Parameter]

property dispersion_center: glotaran.parameter.parameter.Parameter

fill(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

property label: str

property model_dispersion_with_wavenumber: bool

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property normalize: bool

parameter(*global_index: int*, *global_axis: numpy.ndarray*)

property scale: List[glotaran.parameter.parameter.Parameter]

property shift: List[glotaran.parameter.parameter.Parameter]

property type: str

validate(*model: Model*, *parameters=None*) → list[str]

property width: List[glotaran.parameter.parameter.Parameter]

`property width_dispersion: List[glotaran.parameter.parameter.Parameter]`

spectral_matrix

Glotaran Spectral Matrix

spectral_penalties

This package contains compartment constraint items.

Classes

Summary

EqualAreaPenalty	An equal area constraint adds a the differenc of the sum of a compartments in the e matrix in one ore more intervals to the scaled sum of the e matrix of one or more target compartments to residual.

EqualAreaPenalty

`class glotaran.builtin.models.kinetic_spectrum.spectral_penalties.EqualAreaPenalty`

Bases: `object`

An equal area constraint adds a the differenc of the sum of a compartments in the e matrix in one ore more intervals to the scaled sum of the e matrix of one or more target compartments to residual. The additional residual is scaled with the weight.

Attributes Summary

parameter
source
source_intervals
target
target_intervals
weight

parameter

`EqualAreaPenalty.parameter`

source

`EqualAreaPenalty.source`

source_intervals

`EqualAreaPenalty.source_intervals`

target

`EqualAreaPenalty.target`

target_intervals

`EqualAreaPenalty.target_intervals`

weight

`EqualAreaPenalty.weight`

Methods Summary

applies	Returns true if the index is in one of the intervals.
fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	
mprint	
validate	

applies

EqualAreaPenalty.**applies**(*index: Any*) → bool

Returns true if the index is in one of the intervals.

Parameters **index** –

Returns applies

Return type bool

fill

EqualAreaPenalty.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

classmethod EqualAreaPenalty.**from_dict**(*values: dict*) → cls

from_list

classmethod EqualAreaPenalty.**from_list**(*values: list*) → cls

mprint

EqualAreaPenalty.**mprint**(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

validate

EqualAreaPenalty.**validate**(*model: Model*, *parameters=None*) → list[str]

Methods Documentation

applies(*index: Any*) → bool

Returns true if the index is in one of the intervals.

Parameters **index** –

Returns applies

Return type bool

fill(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property parameter: glotaran.parameter.parameter.Parameter

property source: str

property source_intervals: List[Tuple[float, float]]

property target: str

property target_intervals: List[Tuple[float, float]]

validate(*model: Model*, *parameters=None*) → list[str]

property weight: str

spectral_relations

Glotaran Spectral Relation

Functions

Summary

apply_spectral_relations
create_spectral_relation_matrix

continues on next page

Table 112 – continued from previous page

retrieve_related_clps

apply_spectral_relations

glotaran.builtin.models.kinetic_spectrum.spectral_relations.**apply_spectral_relations**(*model: KineticSpectrumModel*, *dataset: str*, *parameters: ParameterGroup*, *clp_labels: list[str]*, *matrix: np.ndarray*, *index: float*) → tuple[list[str], np.ndarray]

create_spectral_relation_matrix

`glotaran.builtin.models.kinetic_spectrum.spectral_relations.`**`create_spectral_relation_matrix`**(*model: KineticSpectrumModel, dataset: str, parameters: ParameterGroup, clp_labe[...]: list[str], matrix: np.ndarr[...], index: float*) → tuple[list[s[...] np.ndarr[...]

retrieve_related_clps

`glotaran.builtin.models.kinetic_spectrum.spectral_relations.`**`retrieve_related_clps`**(*model: KineticSpectrumModel*, *parameters: ParameterGroup*, *clp_labels: dict[str, list[str] | list[list[str]]]*, *clps: dict[str, list[np.ndarray]]*, *data: dict[str, xr.Dataset]*) → dict[str, list[np.ndarray]]

Classes

Summary

SpectralRelation

SpectralRelation

class
`glotaran.builtin.models.kinetic_spectrum.spectral_relations.`**`SpectralRelation`**

Bases: `object`

Attributes Summary

compartment	
interval	
parameter	
target	

compartment

`SpectralRelation.`**`compartment`**

interval

`SpectralRelation.`**`interval`**

parameter

`SpectralRelation.`**`parameter`**

target

`SpectralRelation.`**`target`**

Methods Summary

applies	Returns true if the index is in one of the intervals.
fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	

continues on next page

Table 115 – continued from previous page

mprint	
validate	

applies

SpectralRelation.**applies**(*index: Any*) → bool

Returns true if the index is in one of the intervals.

Parameters **index** –

Returns applies

Return type bool

fill

SpectralRelation.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

classmethod SpectralRelation.**from_dict**(*values: dict*) → cls

from_list

classmethod SpectralRelation.**from_list**(*values: list*) → cls

mprint

SpectralRelation.**mprint**(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

validate

SpectralRelation.**validate**(*model: Model*, *parameters=None*) → list[str]

Methods Documentation

applies(*index: Any*) → bool

Returns true if the index is in one of the intervals.

Parameters **index** –

Returns applies

Return type bool

property compartment: str

fill(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

property interval: List[Tuple[float, float]]

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property parameter: glotaran.parameter.parameter.Parameter

property target: str

validate(*model: Model*, *parameters=None*) → list[str]

spectral_shape

This package contains the spectral shape item.

Classes

Summary

SpectralShape	Base class for spectral shapes
SpectralShapeGaussian	A gaussian spectral shape
SpectralShapeOne	A gaussian spectral shape
SpectralShapeZero	A gaussian spectral shape

SpectralShape

class glotaran.builtin.models.kinetic_spectrum.spectral_shape.**SpectralShape**

Bases: object

Base class for spectral shapes

Methods Summary

add_type	
get_default_type	

add_type

classmethod SpectralShape.**add_type**(*type_name: str*, *attribute_type: type*)

get_default_type

classmethod SpectralShape.**get_default_type**() → str

Methods Documentation

classmethod add_type(*type_name: str*, *attribute_type: type*)

classmethod get_default_type() → str

SpectralShapeGaussian

class
glotaran.builtin.models.kinetic_spectrum.spectral_shape.**SpectralShapeGaussian**

Bases: `object`

A gaussian spectral shape

Attributes Summary

amplitude
label
location
type
width

amplitude

SpectralShapeGaussian.**amplitude**

label

SpectralShapeGaussian.**label**

location

SpectralShapeGaussian.**location**

type

SpectralShapeGaussian.**type**

width

SpectralShapeGaussian.**width**

Methods Summary

calculate	calculate calculates the shape.
fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	
mprint	
validate	

calculate

SpectralShapeGaussian.**calculate**(*axis: numpy.ndarray*) → numpy.ndarray

calculate calculates the shape.

Parameters **axis** (*np.ndarray*) – The axis to calculate the shape on.

Returns **shape**

Return type numpy.ndarray

fill

SpectralShapeGaussian.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

classmethod SpectralShapeGaussian.**from_dict**(*values: dict*) → cls

from_list

classmethod SpectralShapeGaussian.**from_list**(*values: list*) → cls

mprint

SpectralShapeGaussian.**mprint**(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

validate

SpectralShapeGaussian.**validate**(*model: Model*, *parameters=None*) → list[str]

Methods Documentation

property amplitude: glotaran.parameter.parameter.Parameter

calculate(*axis: numpy.ndarray*) → numpy.ndarray

calculate calculates the shape.

Parameters **axis** (*np.ndarray*) – The axis to calculate the shape on.

Returns shape

Return type numpy.ndarray

fill(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

property label: str

property location: glotaran.parameter.parameter.Parameter

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property type: str

validate(*model: Model*, *parameters=None*) → list[str]

property width: glotaran.parameter.parameter.Parameter

SpectralShapeOne

class glotaran.builtin.models.kinetic_spectrum.spectral_shape.**SpectralShapeOne**

Bases: `object`

A gaussian spectral shape

Attributes Summary

label	
type	

label

SpectralShapeOne.**label**

type

SpectralShapeOne.**type**

Methods Summary

calculate	calculate calculates the shape.
fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	
mprint	
validate	

calculate

`SpectralShapeOne.calculate(axis: numpy.ndarray) → numpy.ndarray`

calculate calculates the shape.

Parameters **axis** (*np.ndarray*) – The axies to calculate the shape on.

Returns shape

Return type numpy.ndarray

fill

`SpectralShapeOne.fill(model: Model, parameters: ParameterGroup) → cls`

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

`classmethod SpectralShapeOne.from_dict(values: dict) → cls`

from_list

`classmethod SpectralShapeOne.from_list(values: list) → cls`

mprint

`SpectralShapeOne.mprint(parameters: ParameterGroup = None, initial_parameters: ParameterGroup = None) → str`

validate

`SpectralShapeOne.validate(model: Model, parameters=None) → list[str]`

Methods Documentation

calculate(*axis: numpy.ndarray*) → numpy.ndarray

calculate calculates the shape.

Parameters **axis** (*np.ndarray*) – The axies to calculate the shape on.

Returns shape

Return type numpy.ndarray

fill(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

property label: str

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property type: str

validate(*model: Model*, *parameters=None*) → list[str]

SpectralShapeZero

class glotaran.builtin.models.kinetic_spectrum.spectral_shape.**SpectralShapeZero**

Bases: object

A gaussian spectral shape

Attributes Summary

label	
type	

label

`SpectralShapeZero.label`

type

`SpectralShapeZero.type`

Methods Summary

calculate	calculate calculates the shape.
fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	
mprint	
validate	

calculate

`SpectralShapeZero.calculate`(*axis: numpy.ndarray*) → numpy.ndarray

calculate calculates the shape.

Only works after calling calling 'fill'.

Parameters **axis** (*np.ndarray*) – The axies to calculate the shape on.

Returns **shape**

Return type numpy.ndarray

fill

`SpectralShapeZero.fill`(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

classmethod SpectralShapeZero.**from_dict**(*values: dict*) → cls

from_list

classmethod SpectralShapeZero.**from_list**(*values: list*) → cls

mprint

SpectralShapeZero.**mprint**(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

validate

SpectralShapeZero.**validate**(*model: Model*, *parameters=None*) → list[str]

Methods Documentation

calculate(*axis: numpy.ndarray*) → numpy.ndarray

calculate calculates the shape.

Only works after calling calling 'fill'.

Parameters **axis** (*np.ndarray*) – The axies to calculate the shape on.

Returns shape

Return type numpy.ndarray

fill(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod **from_dict**(*values: dict*) → cls

classmethod **from_list**(*values: list*) → cls

property label: str

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property type: str

validate(*model: Model*, *parameters=None*) → list[str]

spectral

Modules

glotaran.builtin.models.spectral.shape	This package contains the spectral shape item.
glotaran.builtin.models.spectral.spectral_megacomplex	
glotaran.builtin.models.spectral.spectral_model	
glotaran.builtin.models.spectral.spectral_result	

shape

This package contains the spectral shape item.

Classes

Summary

SpectralShape	Base class for spectral shapes
SpectralShapeOne	A constant spectral shape with value 1
SpectralShapeSkewedGaussian	A (skewed) Gaussian spectral shape
SpectralShapeZero	A constant spectral shape with value 0

SpectralShape

class glotaran.builtin.models.spectral.shape.**SpectralShape**

Bases: object

Base class for spectral shapes

Methods Summary

add_type
get_default_type

add_type

classmethod SpectralShape.**add_type**(*type_name: str*, *attribute_type: type*)

get_default_type

classmethod SpectralShape.**get_default_type**() → str

Methods Documentation

classmethod add_type(*type_name: str*, *attribute_type: type*)

classmethod get_default_type() → str

SpectralShapeOne

class glotaran.builtin.models.spectral.shape.**SpectralShapeOne**

Bases: `object`

A constant spectral shape with value 1

Attributes Summary

label	
type	

label

SpectralShapeOne.**label**

type

`SpectralShapeOne.type`

Methods Summary

calculate	calculate calculates the shape.
fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	
mprint	
validate	

calculate

`SpectralShapeOne.calculate(axis: numpy.ndarray) → numpy.ndarray`

calculate calculates the shape.

Parameters **axis** (*np.ndarray*) – The axis to calculate the shape on.

Returns **shape**

Return type numpy.ndarray

fill

`SpectralShapeOne.fill(model: Model, parameters: ParameterGroup) → cls`

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

classmethod SpectralShapeOne.**from_dict**(*values: dict*) → cls

from_list

classmethod SpectralShapeOne.**from_list**(*values: list*) → cls

mprint

SpectralShapeOne.**mprint**(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

validate

SpectralShapeOne.**validate**(*model: Model*, *parameters=None*) → list[str]

Methods Documentation

calculate(*axis: numpy.ndarray*) → numpy.ndarray

calculate calculates the shape.

Parameters **axis** (*np.ndarray*) – The axis to calculate the shape on.

Returns shape

Return type numpy.ndarray

fill(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

property label: str

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property type: str

validate(*model: Model*, *parameters=None*) → list[str]

SpectralShapeSkewedGaussian

class glotaran.builtin.models.spectral.shape.**SpectralShapeSkewedGaussian**

Bases: object

A (skewed) Gaussian spectral shape

Attributes Summary

amplitude
label
location
skewness
type
width

amplitude

SpectralShapeSkewedGaussian.**amplitude**

label

SpectralShapeSkewedGaussian.**label**

location

SpectralShapeSkewedGaussian.**location**

skewness

SpectralShapeSkewedGaussian.**skewness**

type

SpectralShapeSkewedGaussian.**type**

width

SpectralShapeSkewedGaussian.**width**

Methods Summary

calculate	Calculate a (skewed) Gaussian shape for a given axis.
calculate_gaussian	Calculate a normal Gaussian shape for a given axis.
calculate_skewed_gaussian	Calculate the skewed Gaussian shape for axis.
fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	
mprint	
validate	

calculate

SpectralShapeSkewedGaussian.**calculate**(*axis: numpy.ndarray*) → numpy.ndarray

Calculate a (skewed) Gaussian shape for a given `axis`.

If a non-zero `skewness` parameter was added *calculate_skewed_gaussian()* will be used. Otherwise it will use *calculate_gaussian()*.

Parameters **axis** (*np.ndarray*) – The axis to calculate the shape for.

Returns **shape** – A Gaussian shape.

Return type numpy.ndarray

See also:

calculate_gaussian, *calculate_skewed_gaussian*

Note: Internally `axis` is converted from nm to 1/cm, thus `location` and `width` also need to be provided in 1/cm (`1e7/value_in_nm`).

calculate_gaussian

`SpectralShapeSkewedGaussian.`**`calculate_gaussian`**`(`*axis: numpy.ndarray*`)` → numpy.ndarray

Calculate a normal Gaussian shape for a given `axis`.

The following equation is used for the calculation:

$$f(x, A, x_0, \Delta) = A \exp \left(-\frac{\log(2) \left(2(x - x_0) \right)^2}{\Delta^2} \right)$$

The parameters of the equation represent the following attributes of the shape:

- x : `axis`
- A : `amplitude`
- x_0 : `location`
- Δ : `width`

In this formalism, Δ represents the full width at half maximum (FWHM). Compared to the more common definition $\exp\left(-(x-\mu)^2/(2\sigma^2)\right)$ we have $\sigma = \Delta/(2\sqrt{2\ln(2)}) = \Delta/2.35482$

Parameters **`axis`** (*`np.ndarray`*) – The axis to calculate the shape for.

Returns An array representing a Gaussian shape.

Return type np.ndarray

calculate_skewed_gaussian

`SpectralShapeSkewedGaussian.`**`calculate_skewed_gaussian`**`(`*axis: numpy.ndarray*`)` → numpy.ndarray

Calculate the skewed Gaussian shape for `axis`.

The following equation is used for the calculation:

$$f(x, x_0, A, \Delta, b) = \begin{cases} 0 & \text{if } \theta \leq 0 \\ A \exp \left(-\frac{\log(2) \log\left(\theta(x, x_0, \Delta, b)\right)^2}{b^2} \right) & \text{if } \theta > 0 \end{cases}$$

With:

$$\theta(x, x_0, \Delta, b) = \frac{2b\,(x - x_0) + \Delta}{\Delta}$$

The parameters of the equation represent the following attributes of the shape:

- x : `axis`
- A : `amplitude`
- x_0 : `location`
- Δ : `width`
- b : `skewness`

Where Δ represents the full width at half maximum (FWHM), see `calculate_gaussian()`.

Note that in the limit of skewness parameter b equal to zero $f(x, x_0, A, \Delta, b)$ simplifies to a normal gaussian (since $\lim_{b\to 0} \frac{\ln(1+bx)}{b} = x$), see the definition in `calculate_gaussian()`.

Parameters **axis** (*np.ndarray*) – The axis to calculate the shape for.

Returns An array representing a skewed Gaussian shape.

Return type np.ndarray

fill

`SpectralShapeSkewedGaussian.fill(model: Model, parameters: ParameterGroup)` → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

`classmethod SpectralShapeSkewedGaussian.from_dict(values: dict)` → cls

from_list

`classmethod SpectralShapeSkewedGaussian.from_list(values: list)` → cls

mprint

`SpectralShapeSkewedGaussian.mprint(parameters: ParameterGroup = None, initial_parameters: ParameterGroup = None)` → str

validate

`SpectralShapeSkewedGaussian.validate(model: Model, parameters=None)` → list[str]

Methods Documentation

property amplitude: glotaran.parameter.parameter.Parameter

calculate(*axis: numpy.ndarray*) → numpy.ndarray

Calculate a (skewed) Gaussian shape for a given `axis`.

If a non-zero `skewness` parameter was added `calculate_skewed_gaussian()` will be used. Otherwise it will use `calculate_gaussian()`.

Parameters **axis** (*np.ndarray*) – The axis to calculate the shape for.

Returns **shape** – A Gaussian shape.

Return type numpy.ndarray

See also:

`calculate_gaussian`, `calculate_skewed_gaussian`

Note: Internally `axis` is converted from nm to $1/\text{cm}$, thus `location` and `width` also need to be provided in $1/\text{cm}$ (`1e7/value_in_nm`).

---

calculate_gaussian(*axis: numpy.ndarray*) → numpy.ndarray

Calculate a normal Gaussian shape for a given `axis`.

The following equation is used for the calculation:

$$f(x, A, x_0, \Delta) = A \exp\left(-\frac{\log(2)\left(2(x - x_0)\right)^2}{\Delta^2}\right)$$

The parameters of the equation represent the following attributes of the shape:

- x : `axis`
- A : `amplitude`
- x_0 : `location`
- Δ : `width`

In this formalism, Δ represents the full width at half maximum (FWHM). Compared to the more common definition $\exp\left(-(x-\mu)^2/(2\sigma^2)\right)$ we have $\sigma = \Delta/(2\sqrt{2\ln(2)}) = \Delta/2.35482$

Parameters **axis** (*np.ndarray*) – The axis to calculate the shape for.

Returns An array representing a Gaussian shape.

Return type np.ndarray

calculate_skewed_gaussian(*axis: numpy.ndarray*) → numpy.ndarray

Calculate the skewed Gaussian shape for `axis`.

The following equation is used for the calculation:

$$f(x, x_0, A, \Delta, b) = \begin{cases} 0 & \text{if } \theta \leq 0 \\ A \exp\left(-\frac{\log(2)\log\left(\theta(x, x_0, \Delta, b)\right)^2}{b^2}\right) & \text{if } \theta > 0 \end{cases}$$

With:

$$\theta(x, x_0, \Delta, b) = \frac{2b\,(x - x_0) + \Delta}{\Delta}$$

The parameters of the equation represent the following attributes of the shape:

- x : `axis`
- A : `amplitude`
- x_0 : `location`
- Δ : `width`
- b : `skewness`

Where Δ represents the full width at half maximum (FWHM), see `calculate_gaussian()`.

Note that in the limit of skewness parameter b equal to zero $f(x, x_0, A, \Delta, b)$ simplifies to a normal gaussian (since $\lim_{b\to 0} \frac{\ln(1+bx)}{b} = x$), see the definition in `calculate_gaussian()`.

Parameters **axis** (*np.ndarray*) – The axis to calculate the shape for.

Returns An array representing a skewed Gaussian shape.

Return type np.ndarray

fill(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (`ParameterGroup`) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

property label: str

property location: glotaran.parameter.parameter.Parameter

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property skewness: glotaran.parameter.parameter.Parameter

property type: str

validate(*model: Model*, *parameters=None*) → list[str]

property width: glotaran.parameter.parameter.Parameter

SpectralShapeZero

class glotaran.builtin.models.spectral.shape.**SpectralShapeZero**

Bases: object

A constant spectral shape with value 0

Attributes Summary

label	
type	

label

SpectralShapeZero.**label**

type

SpectralShapeZero.**type**

Methods Summary

calculate	calculate calculates the shape.
fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	
mprint	
validate	

calculate

SpectralShapeZero.**calculate**(*axis: numpy.ndarray*) → numpy.ndarray
calculate calculates the shape.

Only works after calling `fill`.

Parameters **axis** (*np.ndarray*) – The axis to calculate the shape on.

Returns **shape**

Return type numpy.ndarray

fill

SpectralShapeZero.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls
Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

classmethod SpectralShapeZero.**from_dict**(*values: dict*) → cls

from_list

classmethod SpectralShapeZero.**from_list**(*values: list*) → cls

mprint

SpectralShapeZero.**mprint**(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

validate

SpectralShapeZero.**validate**(*model: Model*, *parameters=None*) → list[str]

Methods Documentation

calculate(*axis: numpy.ndarray*) → numpy.ndarray
calculate calculates the shape.

Only works after calling `fill`.

Parameters **axis** (*np.ndarray*) – The axis to calculate the shape on.

Returns shape

Return type numpy.ndarray

fill(*model: Model*, *parameters: ParameterGroup*) → cls
Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (`ParameterGroup`) – The parameter group to fill from.

classmethod **from_dict**(*values: dict*) → cls

classmethod **from_list**(*values: list*) → cls

property **label**: str

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property **type**: str

validate(*model: Model*, *parameters=None*) → list[str]

spectral_megacomplex

Classes

Summary

SpectralMegacomplex	

SpectralMegacomplex

class glotaran.builtin.models.spectral.spectral_megacomplex.**SpectralMegacomplex**

Bases: *glotaran.model.megacomplex.Megacomplex*

Attributes Summary

label	
shape	
type	

label

SpectralMegacomplex.**label**

shape

SpectralMegacomplex.**shape**

type

SpectralMegacomplex.**type**

Methods Summary

calculate_matrix	
fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	
mprint	
validate	

calculate_matrix

`SpectralMegacomplex.calculate_matrix(model, dataset_descriptor: DatasetDescriptor, indices: dict[str, int], axis: dict[str, np.ndarray], **kwargs)`

fill

`SpectralMegacomplex.fill(model: Model, parameters: ParameterGroup) → cls`

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

`classmethod SpectralMegacomplex.from_dict(values: dict) → cls`

from_list

`classmethod SpectralMegacomplex.from_list(values: list) → cls`

mprint

`SpectralMegacomplex.mprint(parameters: ParameterGroup = None, initial_parameters: ParameterGroup = None) → str`

validate

`SpectralMegacomplex.validate(model: Model, parameters=None) → list[str]`

Methods Documentation

calculate_matrix(*model*, *dataset_descriptor: DatasetDescriptor*, *indices: dict[str, int]*, *axis: dict[str, np.ndarray]*, ***kwargs*)

fill(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

property label: str

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property shape: Dict[str, str]

property type: str

validate(*model: Model*, *parameters=None*) → list[str]

spectral_model

Classes

Summary

SpectralModel

SpectralModel

class glotaran.builtin.models.spectral.spectral_model.**SpectralModel**

Bases: *glotaran.model.base_model.Model*

Attributes Summary

additional_penalty_function	
constrain_matrix_function	
dataset	
global_dimension	
global_matrix	
has_additional_penalty_function	
has_matrix_constraints_function	
megacomplex	
model_dimension	
model_type	The type of the model as human readable string.
retrieve_clp_function	
shape	
weights	

additional_penalty_function

SpectralModel.**additional_penalty_function = None**

constrain_matrix_function

SpectralModel.**constrain_matrix_function = None**

dataset

`SpectralModel.dataset`

global_dimension

`SpectralModel.global_dimension = 'time'`

global_matrix

`SpectralModel.global_matrix = None`

has_additional_penalty_function

`SpectralModel.has_additional_penalty_function = None`

has_matrix_constraints_function

`SpectralModel.has_matrix_constraints_function = None`

megacomplex

`SpectralModel.megacomplex`

model_dimension

`SpectralModel.model_dimension = 'spectral'`

model_type

`SpectralModel.model_type`

The type of the model as human readable string.

retrieve_clp_function

`SpectralModel.retrieve_clp_function = None`

shape

SpectralModel.**shape**

weights

SpectralModel.**weights**

Methods Summary

add_weights	
finalize_data	
from_dict	Creates a model from a dictionary.
get_dataset	
get_megacomplex	
get_shape	
grouped	
index_dependent	
markdown	Formats the model as Markdown string.
problem_list	Returns a list with all problems in the model and missing parameters if specified.
set_dataset	
set_megacomplex	
set_shape	
simulate	Simulates the model.
valid	Returns *True* if the number problems in the model is 0, else *False*
validate	Returns a string listing all problems in the model and missing parameters if specified.

add_weights

SpectralModel.**add_weights**(*item: glotaran.model.weight.Weight*)

finalize_data

SpectralModel.**finalize_data**(*problem: Problem, data: dict[str, xr.Dataset]*)

from_dict

classmethod SpectralModel.**from_dict**(*model_dict_ref: dict*) → *glotaran.model.base_model.Model*

Creates a model from a dictionary.

Parameters **model_dict** – Dictionary containing the model.

get_dataset

SpectralModel.**get_dataset**(*label*) → *glotaran.builtin.models.kinetic_image.kinetic_image_dataset_descriptor.KineticImageDa*

get_megacomplex

SpectralModel.**get_megacomplex**(*label*) → glotaran.model.decorator._set_megacomplexes.<locals>.MetaMegacomplex

get_shape

SpectralModel.**get_shape**(*label*) → *glotaran.builtin.models.spectral.shape.SpectralShape*

grouped

SpectralModel.**grouped**()

index_dependent

`SpectralModel.index_dependent()`

markdown

SpectralModel.**markdown**(*parameters: Optional[*glotaran.parameter.parameter_group.ParameterGroup*] = None*, *initial_parameters: Optional[*glotaran.parameter.parameter_group.ParameterGroup*] = None*, *base_heading_level:* int *= 1*) → *glotaran.utils.ipython.MarkdownStr*

Formats the model as Markdown string.

Parameters will be included if specified.

Parameters

- **parameter** (ParameterGroup) – Parameter to include.
- **initial_parameters** (ParameterGroup) – Initial values for the parameters.
- **base_heading_level** (*int*) – Base heading level of the markdown sections.

  E.g.:

  – If it is 1 the string will start with '# Model'.

  – If it is 3 the string will start with '### Model'.

problem_list

SpectralModel.**problem_list**(*parameters: ParameterGroup = None*) → list[str]

Returns a list with all problems in the model and missing parameters if specified.

Parameters **parameter** – The parameter to validate.

set_dataset

SpectralModel.**set_dataset**(*label*, *item:* glotaran.builtin.models.kinetic_image.kinetic_image_dataset_descriptor.KineticImageDa

set_megacomplex

SpectralModel.**set_megacomplex**(*label*, *item: glotaran.model.decorator._set_megacomplexes.<locals>.MetaMegacomplex*)

set_shape

SpectralModel.**set_shape**(*label*, *item:* glotaran.builtin.models.spectral.shape.SpectralShape)

simulate

SpectralModel.**simulate**(*dataset: str*, *parameters: ParameterGroup*, *axes: dict[str, np.ndarray] = None*, *clp: np.ndarray | xr.DataArray = None*, *noise: bool = False*, *noise_std_dev: float = 1.0*, *noise_seed: int = None*) → xr.Dataset

Simulates the model.

Parameters

- **dataset** – Label of the dataset to simulate.
- **parameter** – The parameters for the simulation.
- **axes** – A dictionary with axes for simulation.
- **clp** – Conditionally linear parameters. Used instead of *model.global_matrix* if provided.
- **noise** – If *True* noise is added to the simulated data.
- **noise_std_dev** – The standard deviation of the noise.
- **noise_seed** – Seed for the noise.

valid

SpectralModel.**valid**(*parameters: Optional[*glotaran.parameter.parameter_group.ParameterGroup*] = None*) → bool

Returns *True* if the number problems in the model is 0, else *False*

Parameters **parameter** – The parameter to validate.

validate

`SpectralModel.validate(parameters: Optional[glotaran.parameter.parameter_group.ParameterGroup] = None) → str`

Returns a string listing all problems in the model and missing parameters if specified.

Parameters `parameter` – The parameter to validate.

Methods Documentation

`add_weights(item: glotaran.model.weight.Weight)`

`additional_penalty_function = None`

`constrain_matrix_function = None`

`property dataset: Dict[str, glotaran.builtin.models.kinetic_image.kinetic_image_dataset_descriptor.KineticImageDatasetDescriptor]`

`finalize_data(problem: Problem, data: dict[str, xr.Dataset])`

`classmethod from_dict(model_dict_ref: dict) → glotaran.model.base_model.Model`

Creates a model from a dictionary.

Parameters `model_dict` – Dictionary containing the model.

`get_dataset(label) → glotaran.builtin.models.kinetic_image.kinetic_image_dataset_descriptor.KineticImageDatasetDescriptor`

`get_megacomplex(label) → glotaran.model.decorator._set_megacomplexes.<locals>.MetaMegacomplex`

`get_shape(label) → glotaran.builtin.models.spectral.shape.SpectralShape`

`global_dimension = 'time'`

`global_matrix = None`

`grouped()`

`has_additional_penalty_function = None`

`has_matrix_constraints_function = None`

`index_dependent()`

`markdown(parameters: Optional[glotaran.parameter.parameter_group.ParameterGroup] = None, initial_parameters: Optional[glotaran.parameter.parameter_group.ParameterGroup] = None, base_heading_level: int = 1) → glotaran.utils.ipython.MarkdownStr`

Formats the model as Markdown string.

Parameters will be included if specified.

Parameters

- **parameter** (ParameterGroup) – Parameter to include.
- **initial_parameters** (ParameterGroup) – Initial values for the parameters.
- **base_heading_level** (*int*) – Base heading level of the markdown sections.

  E.g.:

  – If it is 1 the string will start with '# Model'.

  – If it is 3 the string will start with '### Model'.

property megacomplex: Dict[str, glotaran.model.decorator._set_megacomplexes.<locals>.MetaMegacomplex]

model_dimension = 'spectral'

property model_type: str
The type of the model as human readable string.

problem_list(*parameters: ParameterGroup = None*) → list[str]
Returns a list with all problems in the model and missing parameters if specified.

Parameters **parameter** – The parameter to validate.

retrieve_clp_function = None

set_dataset(*label*, *item:* glotaran.builtin.models.kinetic_image.kinetic_image_dataset_descriptor.KineticImageDatasetDescriptor)

set_megacomplex(*label*, *item: glotaran.model.decorator._set_megacomplexes.<locals>.MetaMegacomplex*)

set_shape(*label*, *item:* glotaran.builtin.models.spectral.shape.SpectralShape)

property shape: Dict[str, glotaran.builtin.models.spectral.shape.SpectralShape]

simulate(*dataset: str*, *parameters: ParameterGroup*, *axes: dict[str, np.ndarray] = None*, *clp: np.ndarray | xr.DataArray = None*, *noise: bool = False*, *noise_std_dev: float = 1.0*, *noise_seed: int = None*) → xr.Dataset
Simulates the model.

Parameters

- **dataset** – Label of the dataset to simulate.
- **parameter** – The parameters for the simulation.
- **axes** – A dictionary with axes for simulation.
- **clp** – Conditionally linear parameters. Used instead of *model.global_matrix* if provided.
- **noise** – If *True* noise is added to the simulated data.
- **noise_std_dev** – The standard deviation of the noise.
- **noise_seed** – Seed for the noise.

valid(*parameters: Optional[*glotaran.parameter.parameter_group.ParameterGroup*] = None*) → bool

Returns *True* if the number problems in the model is 0, else *False*

Parameters **parameter** – The parameter to validate.

validate(*parameters: Optional[*glotaran.parameter.parameter_group.ParameterGroup*] = None*) → str

Returns a string listing all problems in the model and missing parameters if specified.

Parameters **parameter** – The parameter to validate.

property weights: Dict[str, glotaran.model.weight.Weight]

spectral_result

Functions

Summary

finalize_spectral_result	
retrieve_spectral_data	

finalize_spectral_result

glotaran.builtin.models.spectral.spectral_result.**finalize_spectral_result**(*model, problem: Problem, data: dict[str, xr.Dataset]*)

retrieve_spectral_data

glotaran.builtin.models.spectral.spectral_result.**retrieve_spectral_data**(*model, dataset, dataset_descriptor*)

12.1.3 cli

Modules

glotaran.cli.commands
glotaran.cli.main

commands

Modules

glotaran.cli.commands.explore
glotaran.cli.commands.export
glotaran.cli.commands.optimize
glotaran.cli.commands.pluginlist
glotaran.cli.commands.print
glotaran.cli.commands.util
glotaran.cli.commands.validate

explore

Functions

Summary

export	Exports data from netCDF4 to ascii.

export

glotaran.cli.commands.explore.**export**(*filename: str*, *select*, *out: str*, *name: str*)

Exports data from netCDF4 to ascii.

export

optimize

Functions

Summary

optimize_cmd	Optimizes a model.

optimize_cmd

glotaran.cli.commands.optimize.**optimize_cmd**(*dataformat: str, data: List[str], out: str, nfev: int, nnls: bool, yes: bool, parameters_file: str, model_file: str, scheme_file: str*)

Optimizes a model. e.g.: glotaran optimize –

pluginlist

Functions

Summary

plugin_list_cmd	Prints a list of installed plugins.

plugin_list_cmd

glotaran.cli.commands.pluginlist.**plugin_list_cmd**()

Prints a list of installed plugins.

print

Functions

Summary

print_cmd	Parses scheme, a model or a parameter file and prints the result as a Markdown formatted string.

print_cmd

glotaran.cli.commands.print.**print_cmd**(*parameters_file: str*, *model_file: str*, *scheme_file: str*)

Parses scheme, a model or a parameter file and prints the result as a Markdown formatted string.

util

Functions

Summary

load_dataset_file
load_model_file
load_parameter_file
load_scheme_file
select_data
select_name
signature_analysis
write_data

load_dataset_file

glotaran.cli.commands.util.**load_dataset_file**(*filename*, *fmt=None*, *verbose=False*)

load_model_file

glotaran.cli.commands.util.**load_model_file**(*filename*, *verbose=False*)

load_parameter_file

glotaran.cli.commands.util.**load_parameter_file**(*filename*, *fmt=None*, *verbose=False*)

load_scheme_file

glotaran.cli.commands.util.**load_scheme_file**(*filename*, *verbose=False*)

select_data

glotaran.cli.commands.util.**select_data**(*data*, *dim*, *selection*)

select_name

glotaran.cli.commands.util.**select_name**(*filename*, *dataset*)

signature_analysis

glotaran.cli.commands.util.**signature_analysis**(*cmd*)

write_data

glotaran.cli.commands.util.**write_data**(*data*, *out*)

Classes

Summary

ValOrRangeOrList

ValOrRangeOrList

class glotaran.cli.commands.util.**ValOrRangeOrList**
Bases: click.types.ParamType

Attributes Summary

arity	
envvar_list_splitter	if a list of this type is expected and the value is pulled from a string environment variable, this is what splits it up.
is_composite	
name	the descriptive name of this type

arity

`ValOrRangeOrList.arity: ClassVar[int] = 1`

envvar_list_splitter

`ValOrRangeOrList.envvar_list_splitter: ClassVar[Optional[str]] = None`

if a list of this type is expected and the value is pulled from a string environment variable, this is what splits it up. *None* means any whitespace. For all parameters the general rule is that whitespace splits them up. The exception are paths and files which are split by `os.path.pathsep` by default (":" on Unix and ";" on Windows).

is_composite

`ValOrRangeOrList.is_composite: ClassVar[bool] = False`

name

`ValOrRangeOrList.name: str = 'number or range or list'`

the descriptive name of this type

Methods Summary

convert	Convert the value to the correct type.
fail	Helper method to fail with an invalid value message.
get_metavar	Returns the metavar default for this param if it provides one.
get_missing_message	Optionally might return extra information about a missing parameter.
shell_complete	Return a list of `CompletionItem` objects for the incomplete value.
split_envvar_value	Given a value from an environment variable this splits it up into small chunks depending on the defined envvar list splitter.

continues on next page

Table 149 – continued from previous page

to_info_dict	Gather information that could be useful for a tool generating user-facing documentation.

convert

`ValOrRangeOrList.convert(value, param, ctx)`

Convert the value to the correct type. This is not called if the value is `None` (the missing value).

This must accept string values from the command line, as well as values that are already the correct type. It may also convert other compatible types.

The `param` and `ctx` arguments may be `None` in certain situations, such as when converting prompt input.

If the value cannot be converted, call *fail()* with a descriptive message.

Parameters

- **value** – The value to convert.
- **param** – The parameter that is using this type to convert its value. May be `None`.
- **ctx** – The current context that arrived at this value. May be `None`.

fail

`ValOrRangeOrList.fail(message: str, param: Optional[Parameter] = None, ctx: Optional[Context] = None) → t.NoReturn`

Helper method to fail with an invalid value message.

get_metavar

`ValOrRangeOrList.get_metavar(param: Parameter) → Optional[str]`

Returns the metavar default for this param if it provides one.

get_missing_message

`ValOrRangeOrList.get_missing_message(param: Parameter) → Optional[str]`

Optionally might return extra information about a missing parameter.

New in version 2.0.

shell_complete

ValOrRangeOrList.**shell_complete**(*ctx: Context*, *param: Parameter*, *incomplete: str*) → List[CompletionItem]

Return a list of CompletionItem objects for the incomplete value. Most types do not provide completions, but some do, and this allows custom types to provide custom completions as well.

Parameters

- **ctx** – Invocation context for this command.
- **param** – The parameter that is requesting completion.
- **incomplete** – Value being completed. May be empty.

New in version 8.0.

split_envvar_value

ValOrRangeOrList.**split_envvar_value**(*rv: str*) → Sequence[str]

Given a value from an environment variable this splits it up into small chunks depending on the defined envvar list splitter.

If the splitter is set to *None*, which means that whitespace splits, then leading and trailing whitespace is ignored. Otherwise, leading and trailing splitters usually lead to empty items being included.

to_info_dict

ValOrRangeOrList.**to_info_dict**() → Dict[str, Any]

Gather information that could be useful for a tool generating user-facing documentation.

Use `click.Context.to_info_dict()` to traverse the entire CLI structure.

New in version 8.0.

Methods Documentation

arity: ClassVar[int] = 1

convert(*value*, *param*, *ctx*)

Convert the value to the correct type. This is not called if the value is `None` (the missing value).

This must accept string values from the command line, as well as values that are already the correct type. It may also convert other compatible types.

The `param` and `ctx` arguments may be `None` in certain situations, such as when converting prompt input.

If the value cannot be converted, call *fail()* with a descriptive message.

Parameters

- **value** – The value to convert.
- **param** – The parameter that is using this type to convert its value. May be `None`.
- **ctx** – The current context that arrived at this value. May be `None`.

envvar_list_splitter: ClassVar[Optional[str]] = None
if a list of this type is expected and the value is pulled from a string environment variable, this is what splits it up. *None* means any whitespace. For all parameters the general rule is that whitespace splits them up. The exception are paths and files which are split by `os.path.pathsep` by default (":" on Unix and ";" on Windows).

fail(*message: str*, *param: Optional[Parameter] = None*, *ctx: Optional[Context] = None*) → t.NoReturn
Helper method to fail with an invalid value message.

get_metavar(*param: Parameter*) → Optional[str]
Returns the metavar default for this param if it provides one.

get_missing_message(*param: Parameter*) → Optional[str]
Optionally might return extra information about a missing parameter.

New in version 2.0.

is_composite: ClassVar[bool] = False

name: str = 'number or range or list'
the descriptive name of this type

shell_complete(*ctx: Context*, *param: Parameter*, *incomplete: str*) → List[CompletionItem]
Return a list of `CompletionItem` objects for the incomplete value. Most types do not provide completions, but some do, and this allows custom types to provide custom completions as well.

Parameters

- **ctx** – Invocation context for this command.
- **param** – The parameter that is requesting completion.
- **incomplete** – Value being completed. May be empty.

New in version 8.0.

split_envvar_value(*rv: str*) → Sequence[str]
Given a value from an environment variable this splits it up into small chunks depending on the defined envvar list splitter.

If the splitter is set to *None*, which means that whitespace splits, then leading and trailing whitespace is ignored. Otherwise, leading and trailing splitters usually lead to empty items being included.

to_info_dict() → Dict[str, Any]
Gather information that could be useful for a tool generating user-facing documentation.

Use `click.Context.to_info_dict()` to traverse the entire CLI structure.

New in version 8.0.

validate

Functions

Summary

validate_cmd	Validates a model file and optionally a parameter file.

validate_cmd

glotaran.cli.commands.validate.**validate_cmd**(*parameters_file: str*, *model_file: str*, *scheme_file: str*)

Validates a model file and optionally a parameter file.

main

Classes

Summary

Cli

Cli

class glotaran.cli.main.**Cli**(**args*, ***kwargs*)

Bases: `click.core.Group`

Attributes Summary

allow_extra_args	the default for the `Context.allow_extra_args` flag.
allow_interspersed_args	the default for the `Context.allow_interspersed_args` flag.
command_class	If set, this is used by the group's *command()* decorator as the default `Command` class.
group_class	If set, this is used by the group's *group()* decorator as the default `Group` class.
ignore_unknown_options	the default for the `Context.ignore_unknown_options` flag.

allow_extra_args

`Cli.allow_extra_args = True`

the default for the `Context.allow_extra_args` flag.

allow_interspersed_args

`Cli.allow_interspersed_args = False`

the default for the `Context.allow_interspersed_args` flag.

command_class

`Cli.command_class: Optional[Type[click.core.Command]] = None`

If set, this is used by the group's *command()* decorator as the default `Command` class. This is useful to make all subcommands use a custom command class.

New in version 8.0.

group_class

`Cli.group_class: Optional[Union[Type[Group], Type[type]]] = None`

If set, this is used by the group's *group()* decorator as the default `Group` class. This is useful to make all subgroups use a custom group class.

If set to the special value `type` (literally `group_class = type`), this group's class will be used as the default class. This makes a custom group class continue to make custom groups.

New in version 8.0.

ignore_unknown_options

`Cli.ignore_unknown_options = False`

the default for the `Context.ignore_unknown_options` flag.

Methods Summary

add_command	Registers another `Command` with this group.
collect_usage_pieces	Returns all the pieces that go into the usage line and returns it as a list of strings.
command	Behaves the same as *click.Group.command()* except capture a priority for listing command names in help.
format_commands	Extra format methods for multi methods that adds all the commands after the options.
format_epilog	Writes the epilog into the formatter if it exists.
format_help	Writes the help into the formatter if it exists.
format_help_text	Writes the help text to the formatter if it exists.

continues on next page

Table 153 – continued from previous page

`format_options`	Writes all the options into the formatter if they exist.
`format_usage`	Writes the usage line into the formatter.
`get_command`	Given a context and a command name, this returns a `Command` object if it exists or returns *None*.
`get_help`	Formats the help into a string and returns it.
`get_help_option`	Returns the help option object.
`get_help_option_names`	Returns the names for the help option.
`get_params`	
`get_short_help_str`	Gets short help for the command or makes it by shortening the long help string.
`get_usage`	Formats the usage line into a string and returns it.
`group`	A shortcut decorator for declaring and attaching a group to the group.
`invoke`	Given a context, this invokes the attached callback (if it exists) in the right way.
`list_commands`	Returns a list of subcommand names in the order they should appear.
`list_commands_for_help`	reorder the list of commands when listing the help
`main`	This is the way to invoke a script with all the bells and whistles as a command line application.
`make_context`	This function when given an info name and arguments will kick off the parsing and create a new `Context`.
`make_parser`	Creates the underlying option parser for this command.
`parse_args`	Given a context and a list of arguments this creates the parser and parses the arguments, then modifies the context as necessary.
`resolve_command`	
`result_callback`	Adds a result callback to the command.
`resultcallback`	
`shell_complete`	Return a list of completions for the incomplete value.
`to_info_dict`	Gather information that could be useful for a tool generating user-facing documentation.

add_command

`Cli.add_command`(*cmd: click.core.Command*, *name: Optional[str] = None*) → None
Registers another `Command` with this group. If the name is not provided, the name of the command is used.

collect_usage_pieces

`Cli.collect_usage_pieces`(*ctx: click.core.Context*) → List[str]
Returns all the pieces that go into the usage line and returns it as a list of strings.

command

`Cli.command`(**args*, ***kwargs*)
Behaves the same as *click.Group.command()* except capture a priority for listing command names in help.

format_commands

`Cli.format_commands`(*ctx: click.core.Context*, *formatter: click.formatting.HelpFormatter*) → None
Extra format methods for multi methods that adds all the commands after the options.

format_epilog

`Cli.format_epilog`(*ctx: click.core.Context*, *formatter: click.formatting.HelpFormatter*) → None
Writes the epilog into the formatter if it exists.

format_help

`Cli.format_help`(*ctx: click.core.Context*, *formatter: click.formatting.HelpFormatter*) → None
Writes the help into the formatter if it exists.

This is a low-level method called by `get_help()`.

This calls the following methods:

- `format_usage()`
- `format_help_text()`
- `format_options()`
- `format_epilog()`

format_help_text

`Cli.format_help_text`(*ctx: click.core.Context*, *formatter: click.formatting.HelpFormatter*) → None

Writes the help text to the formatter if it exists.

format_options

`Cli.format_options`(*ctx: click.core.Context*, *formatter: click.formatting.HelpFormatter*) → None

Writes all the options into the formatter if they exist.

format_usage

`Cli.format_usage`(*ctx: click.core.Context*, *formatter: click.formatting.HelpFormatter*) → None

Writes the usage line into the formatter.

This is a low-level method called by `get_usage()`.

get_command

`Cli.get_command`(*ctx: click.core.Context*, *cmd_name: str*) → Optional[click.core.Command]

Given a context and a command name, this returns a `Command` object if it exists or returns *None*.

get_help

`Cli.get_help`(*ctx*)

Formats the help into a string and returns it.

Calls `format_help()` internally.

get_help_option

`Cli.get_help_option`(*ctx: click.core.Context*) → Optional[click.core.Option]

Returns the help option object.

get_help_option_names

`Cli.get_help_option_names`(*ctx: click.core.Context*) → List[str]

Returns the names for the help option.

get_params

Cli.**get_params**(*ctx: click.core.Context*) → List[click.core.Parameter]

get_short_help_str

Cli.**get_short_help_str**(*limit: int = 45*) → str
Gets short help for the command or makes it by shortening the long help string.

get_usage

Cli.**get_usage**(*ctx: click.core.Context*) → str
Formats the usage line into a string and returns it.

Calls `format_usage()` internally.

group

Cli.**group**(**args: Any*, ***kwargs: Any*) → Callable[[Callable[[...], Any]], click.core.Group]
A shortcut decorator for declaring and attaching a group to the group. This takes the same arguments as `group()` and immediately registers the created group with this group by calling `add_command()`.

To customize the group class used, set the `group_class` attribute.

Changed in version 8.0: Added the `group_class` attribute.

invoke

Cli.**invoke**(*ctx: click.core.Context*) → Any
Given a context, this invokes the attached callback (if it exists) in the right way.

list_commands

Cli.**list_commands**(*ctx: click.core.Context*) → List[str]
Returns a list of subcommand names in the order they should appear.

list_commands_for_help

Cli.**list_commands_for_help**(*ctx*)
reorder the list of commands when listing the help

main

`Cli.main`(*args: Optional[Sequence[str]] = None*, *prog_name: Optional[str] = None*, *complete_var: Optional[str] = None*, *standalone_mode: bool = True*, *windows_expand_args: bool = True*, ***extra: Any*) → Any

This is the way to invoke a script with all the bells and whistles as a command line application. This will always terminate the application after a call. If this is not wanted, `SystemExit` needs to be caught.

This method is also available by directly calling the instance of a `Command`.

Parameters

- **args** – the arguments that should be used for parsing. If not provided, `sys.argv[1:]` is used.
- **prog_name** – the program name that should be used. By default the program name is constructed by taking the file name from `sys.argv[0]`.
- **complete_var** – the environment variable that controls the bash completion support. The default is `"_<prog_name>_COMPLETE"` with prog_name in uppercase.
- **standalone_mode** – the default behavior is to invoke the script in standalone mode. Click will then handle exceptions and convert them into error messages and the function will never return but shut down the interpreter. If this is set to *False* they will be propagated to the caller and the return value of this function is the return value of `invoke()`.
- **windows_expand_args** – Expand glob patterns, user dir, and env vars in command line args on Windows.
- **extra** – extra keyword arguments are forwarded to the context constructor. See `Context` for more information.

Changed in version 8.0.1: Added the `windows_expand_args` parameter to allow disabling command line arg expansion on Windows.

Changed in version 8.0: When taking arguments from `sys.argv` on Windows, glob patterns, user dir, and env vars are expanded.

Changed in version 3.0: Added the `standalone_mode` parameter.

make_context

`Cli.make_context`(*info_name: Optional[str]*, *args: List[str]*, *parent: Optional[click.core.Context] = None*, ***extra: Any*) → click.core.Context

This function when given an info name and arguments will kick off the parsing and create a new `Context`. It does not invoke the actual command callback though.

To quickly customize the context class used without overriding this method, set the `context_class` attribute.

Parameters

- **info_name** – the info name for this invocation. Generally this is the most descriptive name for the script or command. For the toplevel script it's usually the name of the script, for commands below it it's the name of the command.
- **args** – the arguments to parse as list of strings.

- **parent** – the parent context if available.
- **extra** – extra keyword arguments forwarded to the context constructor.

Changed in version 8.0: Added the `context_class` attribute.

make_parser

`Cli.make_parser`(*ctx: click.core.Context*) → click.parser.OptionParser
Creates the underlying option parser for this command.

parse_args

`Cli.parse_args`(*ctx: click.core.Context*, *args: List[str]*) → List[str]
Given a context and a list of arguments this creates the parser and parses the arguments, then modifies the context as necessary. This is automatically invoked by `make_context()`.

resolve_command

`Cli.resolve_command`(*ctx: click.core.Context*, *args: List[str]*) → Tuple[Optional[str], Optional[click.core.Command], List[str]]

result_callback

`Cli.result_callback`(*replace: bool = False*) → Callable[[click.core.F], click.core.F]
Adds a result callback to the command. By default if a result callback is already registered this will chain them but this can be disabled with the *replace* parameter. The result callback is invoked with the return value of the subcommand (or the list of return values from all subcommands if chaining is enabled) as well as the parameters as they would be passed to the main callback.

Example:

```
@click.group()
@click.option('-i', '--input', default=23)
def cli(input):
    return 42

@cli.result_callback()
def process_result(result, input):
    return result + input
```

Parameters **replace** – if set to *True* an already existing result callback will be removed.

Changed in version 8.0: Renamed from `resultcallback`.

New in version 3.0.

resultcallback

`Cli.`**`resultcallback`**(*replace: bool = False*) → Callable[[click.core.F], click.core.F]

shell_complete

`Cli.`**`shell_complete`**(*ctx: click.core.Context*, *incomplete: str*) → List[CompletionItem]

Return a list of completions for the incomplete value. Looks at the names of options, subcommands, and chained multi-commands.

Parameters

- **`ctx`** – Invocation context for this command.
- **`incomplete`** – Value being completed. May be empty.

New in version 8.0.

to_info_dict

`Cli.`**`to_info_dict`**(*ctx: click.core.Context*) → Dict[str, Any]

Gather information that could be useful for a tool generating user-facing documentation. This traverses the entire structure below this command.

Use `click.Context.to_info_dict()` to traverse the entire CLI structure.

Parameters **`ctx`** – A `Context` representing this command.

New in version 8.0.

Methods Documentation

`add_command`(*cmd: click.core.Command*, *name: Optional[str] = None*) → None

Registers another `Command` with this group. If the name is not provided, the name of the command is used.

`allow_extra_args = True`

the default for the `Context.allow_extra_args` flag.

`allow_interspersed_args = False`

the default for the `Context.allow_interspersed_args` flag.

`callback`

the callback to execute when the command fires. This might be *None* in which case nothing happens.

`collect_usage_pieces`(*ctx: click.core.Context*) → List[str]

Returns all the pieces that go into the usage line and returns it as a list of strings.

`command`(**args*, ***kwargs*)

Behaves the same as *click.Group.command()* except capture a priority for listing command names in help.

`command_class: Optional[Type[click.core.Command]] = None`

If set, this is used by the group's `command()` decorator as the default `Command` class. This is useful to make all subcommands use a custom command class.

New in version 8.0.

commands: t.Dict[str, Command]
The registered subcommands by their exported names.

context_class
alias of `click.core.Context`

context_settings: t.Dict[str, t.Any]
an optional dictionary with defaults passed to the context.

format_commands(*ctx: click.core.Context*, *formatter: click.formatting.HelpFormatter*) → None
Extra format methods for multi methods that adds all the commands after the options.

format_epilog(*ctx: click.core.Context*, *formatter: click.formatting.HelpFormatter*) → None
Writes the epilog into the formatter if it exists.

format_help(*ctx: click.core.Context*, *formatter: click.formatting.HelpFormatter*) → None
Writes the help into the formatter if it exists.

This is a low-level method called by `get_help()`.

This calls the following methods:

- `format_usage()`
- `format_help_text()`
- `format_options()`
- `format_epilog()`

format_help_text(*ctx: click.core.Context*, *formatter: click.formatting.HelpFormatter*) → None
Writes the help text to the formatter if it exists.

format_options(*ctx: click.core.Context*, *formatter: click.formatting.HelpFormatter*) → None
Writes all the options into the formatter if they exist.

format_usage(*ctx: click.core.Context*, *formatter: click.formatting.HelpFormatter*) → None
Writes the usage line into the formatter.

This is a low-level method called by `get_usage()`.

get_command(*ctx: click.core.Context*, *cmd_name: str*) → Optional[click.core.Command]
Given a context and a command name, this returns a `Command` object if it exists or returns *None*.

get_help(*ctx*)
Formats the help into a string and returns it.

Calls `format_help()` internally.

get_help_option(*ctx: click.core.Context*) → Optional[click.core.Option]
Returns the help option object.

get_help_option_names(*ctx: click.core.Context*) → List[str]
Returns the names for the help option.

get_params(*ctx: click.core.Context*) → List[click.core.Parameter]

get_short_help_str(*limit: int = 45*) → str
Gets short help for the command or makes it by shortening the long help string.

get_usage(*ctx: click.core.Context*) → str
Formats the usage line into a string and returns it.

Calls *format_usage()* internally.

group(**args: Any*, ***kwargs: Any*) → Callable[[Callable[[...], Any]], click.core.Group]
A shortcut decorator for declaring and attaching a group to the group. This takes the same arguments as *group()* and immediately registers the created group with this group by calling *add_command()*.

To customize the group class used, set the *group_class* attribute.

Changed in version 8.0: Added the *group_class* attribute.

group_class: Optional[Union[Type[Group], Type[type]]] = None
If set, this is used by the group's *group()* decorator as the default `Group` class. This is useful to make all subgroups use a custom group class.

If set to the special value `type` (literally `group_class = type`), this group's class will be used as the default class. This makes a custom group class continue to make custom groups.

New in version 8.0.

ignore_unknown_options = False
the default for the `Context.ignore_unknown_options` flag.

invoke(*ctx: click.core.Context*) → Any
Given a context, this invokes the attached callback (if it exists) in the right way.

list_commands(*ctx: click.core.Context*) → List[str]
Returns a list of subcommand names in the order they should appear.

list_commands_for_help(*ctx*)
reorder the list of commands when listing the help

main(*args: Optional[Sequence[str]] = None*, *prog_name: Optional[str] = None*, *complete_var: Optional[str] = None*, *standalone_mode: bool = True*, *windows_expand_args: bool = True*, ***extra: Any*) → Any
This is the way to invoke a script with all the bells and whistles as a command line application. This will always terminate the application after a call. If this is not wanted, `SystemExit` needs to be caught.

This method is also available by directly calling the instance of a `Command`.

Parameters

- **args** – the arguments that should be used for parsing. If not provided, `sys.argv[1:]` is used.
- **prog_name** – the program name that should be used. By default the program name is constructed by taking the file name from `sys.argv[0]`.
- **complete_var** – the environment variable that controls the bash completion support. The default is `"_<prog_name>_COMPLETE"` with prog_name in uppercase.
- **standalone_mode** – the default behavior is to invoke the script in standalone mode. Click will then handle exceptions and convert them into error messages and the function will never return but shut down the interpreter. If this is set to *False* they will be propagated to the caller and the return value of this function is the return value of *invoke()*.

- **windows_expand_args** – Expand glob patterns, user dir, and env vars in command line args on Windows.
- **extra** – extra keyword arguments are forwarded to the context constructor. See `Context` for more information.

Changed in version 8.0.1: Added the `windows_expand_args` parameter to allow disabling command line arg expansion on Windows.

Changed in version 8.0: When taking arguments from `sys.argv` on Windows, glob patterns, user dir, and env vars are expanded.

Changed in version 3.0: Added the `standalone_mode` parameter.

make_context(*info_name: Optional[str], args: List[str], parent: Optional[click.core.Context] = None, **extra: Any*) → click.core.Context

This function when given an info name and arguments will kick off the parsing and create a new `Context`. It does not invoke the actual command callback though.

To quickly customize the context class used without overriding this method, set the `context_class` attribute.

Parameters

- **info_name** – the info name for this invocation. Generally this is the most descriptive name for the script or command. For the toplevel script it's usually the name of the script, for commands below it it's the name of the command.
- **args** – the arguments to parse as list of strings.
- **parent** – the parent context if available.
- **extra** – extra keyword arguments forwarded to the context constructor.

Changed in version 8.0: Added the `context_class` attribute.

make_parser(*ctx: click.core.Context*) → click.parser.OptionParser

Creates the underlying option parser for this command.

name

the name the command thinks it has. Upon registering a command on a `Group` the group will default the command name with this information. You should instead use the `Context`'s `info_name` attribute.

params: t.List[Parameter]

the list of parameters for this command in the order they should show up in the help page and execute. Eager parameters will automatically be handled before non eager ones.

parse_args(*ctx: click.core.Context, args: List[str]*) → List[str]

Given a context and a list of arguments this creates the parser and parses the arguments, then modifies the context as necessary. This is automatically invoked by `make_context()`.

resolve_command(*ctx: click.core.Context, args: List[str]*) → Tuple[Optional[str], Optional[click.core.Command], List[str]]

result_callback(*replace: bool = False*) → Callable[[click.core.F], click.core.F]

Adds a result callback to the command. By default if a result callback is already registered this will chain them but this can be disabled with the *replace* parameter. The result callback is invoked with the return value of the subcommand (or the list of return values from all subcommands if chaining is enabled) as well as the parameters as they would be passed to the main callback.

Example:

```
@click.group()
@click.option('-i', '--input', default=23)
def cli(input):
    return 42

@cli.result_callback()
def process_result(result, input):
    return result + input
```

Parameters **replace** – if set to *True* an already existing result callback will be removed.

Changed in version 8.0: Renamed from `resultcallback`.

New in version 3.0.

resultcallback(*replace: bool = False*) → Callable[[click.core.F], click.core.F]

shell_complete(*ctx: click.core.Context*, *incomplete: str*) → List[CompletionItem]

Return a list of completions for the incomplete value. Looks at the names of options, subcommands, and chained multi-commands.

Parameters

- **ctx** – Invocation context for this command.
- **incomplete** – Value being completed. May be empty.

New in version 8.0.

to_info_dict(*ctx: click.core.Context*) → Dict[str, Any]

Gather information that could be useful for a tool generating user-facing documentation. This traverses the entire structure below this command.

Use `click.Context.to_info_dict()` to traverse the entire CLI structure.

Parameters **ctx** – A `Context` representing this command.

New in version 8.0.

12.1.4 deprecation

Deprecation helpers and place to put deprecated implementations till removing.

Modules

glotaran.deprecation.deprecation_utils	Helper functions to give deprecation warnings.
glotaran.deprecation.modules	Package containing deprecated implementations which were removed.

deprecation_utils

Helper functions to give deprecation warnings.

Functions

Summary

check_qualnames_in_tests	Test that qualnames import path exists when running tests.
deprecate	Decorate a function, method or class to deprecate it.
deprecate_module_attribute	Import and return and anttribute from the new location.
deprecate_submodule	Create a module at runtime which retrieves attributes from new module.
glotaran_version	Version of the distribution.
module_attribute	Import and return the attribute (e.g.
parse_version	Parse version string to tuple of three ints for comparison.
warn_deprecated	Raise deprecation warning with change information.

check_qualnames_in_tests

glotaran.deprecation.deprecation_utils.**check_qualnames_in_tests**(*qual_names: Sequence[str], importable_indices: Sequence[int]*)

Test that qualnames import path exists when running tests.

All deprecations should be tested anyway in order to get the proper errors when a deprecation is overdue. This helperfunction also helps to ensure that at least the import paths (`qual_names`) of the old and new usage exist.

Parameters

- **qual_names** (*Sequence[str]*) – Sequence of fully qualified module attribute names, optionally with call arguments.
- **importable_indices** (*Sequence[int]*) – Indices of corresponding to `qual_names` indicating how to slice each `qual_name` split at `.`, for the import and attribute checking.

See also:

warn_deprecated, *deprecate*

deprecate

`glotaran.deprecation.deprecation_utils.deprecate(*, deprecated_qual_name_usage: str, new_qual_name_usage: str, to_be_removed_in_version: str, has_glotaran_replacement: bool = True, importable_indices: tuple[int, int] = (1, 1))` → Callable[[DecoratedCallable], DecoratedCallable]

Decorate a function, method or class to deprecate it.

This raises deprecation warning with old / new usage information and end of support version.

Parameters

- **deprecated_qual_name_usage** (*str*) – Old usage with fully qualified name e.g.: `'glotaran.read_model_from_yaml(model_yml_str)'`
- **new_qual_name_usage** (*str*) – New usage as fully qualified name e.g.: `'glotaran.io.load_model(model_yml_str, format_name="yml_str")'`
- **to_be_removed_in_version** (*str*) – Version the support for this usage will be removed.
- **has_glotaran_replacement** (*bool*) – Whether or not this functionality has a replacement in core pyglotaran. This will be mapped to the second entry of `check_qualnames` in `warn_deprecated()`.
- **importable_indices** (*Sequence[int]*) – Indices from right for most nested item which is importable for `deprecated_qual_name_usage` and `new_qual_name_usage` after splitting at `.`. This is used when the old or new usage is a method or mapping access. E.g. let `deprecated_qual_name_usage` be `package.module.class.mapping["key"]`, then you would use `importable_indices=(2, 1)`, this way func:*check_qualnames_in_tests* will import `package.module.class` and check if `class` has an attribute `mapping`. Default

Returns Original function or class throwing a Deprecation warning when used.

Return type DecoratedCallable

Warns OverDueDeprecation – If the current version is greater or equal to `end_of_life_version`.

See also:

warn_deprecated, *deprecate_module_attribute*, *deprecate_submodule*, *check_qualnames_in_tests*

Examples

This is the way the old `read_parameters_from_yaml_file` was deprecated and the usage of `load_model` was promoted instead.

Listing 1: glotaran/deprecation/modules/glotaran_root.py

```
@deprecate(
    deprecated_qualname_usage="glotaran.read_parameters_from_yaml_
↪file(model_path)",
    new_qualname_usage="glotaran.io.load_model(model_path)",
    to_be_removed_in_version="0.6.0",
)
def read_parameters_from_yaml_file(model_path: str):
    return load_model(model_path)
```

deprecate_module_attribute

`glotaran.deprecation.deprecation_utils.`**`deprecate_module_attribute`**`(*, deprecated_qual_name: str, new_qual_name: str, to_be_removed_in_version: str) → Any`

Import and return and anttribute from the new location.

This needs to be wrapped in the definition of a module wide `__getattr__` function so it won't throw warnings all the time (see example).

Parameters

- **deprecated_qual_name** (*str*) – Fully qualified name of the deprecated attribute e.g.: `glotaran.ParameterGroup`
- **new_qual_name** (*str*) – Fully qualified name of the new attribute e.g.: `glotaran.parameter.ParameterGroup`
- **to_be_removed_in_version** (*str*) – Version the support for this usage will be removed.

Returns Module attribute from its new location.

Return type Any

See also:

deprecate, *warn_deprecated*, *deprecate_submodule*

Examples

When deprecating the usage of `ParameterGroup` the root of `glotaran` and promoting to import it from `glotaran.parameter` the following code was added to the root `__init__.py`.

Listing 2: glotaran/__init__.py

```
def __getattr__(attribute_name: str):
    from glotaran.deprecation import deprecate_module_attribute

    if attribute_name == "ParameterGroup":
        return deprecate_module_attribute(
            deprecated_qual_name="glotaran.ParameterGroup",
            new_qual_name="glotaran.parameter.ParameterGroup",
            to_be_removed_in_version="0.6.0",
        )

    raise AttributeError(f"module {__name__} has no attribute {attribute_
↪name}")
```

deprecate_submodule

`glotaran.deprecation.deprecation_utils.`**`deprecate_submodule`**`(*, deprecated_module_name: str, new_module_name: str, to_be_removed_in_version: str) → module`

Create a module at runtime which retrieves attributes from new module.

When moving a module, create a variable with the modules name in the parent packages `__init__.py`, so imports will be redirected to the new module location and a deprecation warning will be given, to help the user adjust the outdated code. Each time an attribute is retrieved there will be a deprecation warning.

Parameters

- **deprecated_module_name** (*str*) – Fully qualified name of the deprecated module e.g.: `'glotaran.analysis.result'`
- **new_module_name** (*str*) – Fully qualified name of the new module e.g.: `'glotaran.project.result'`
- **to_be_removed_in_version** (*str*) – Version the support for this usage will be removed.

Returns Module containing

Return type ModuleType

See also:

deprecate, *deprecate_module_attribute*

Examples

When moving the module `result` from `glotaran.analysis.result` to `glotaran.project.result` the following code was added to the old parent packages (`glotaran.analysis`) `__init__.py`.

Listing 3: glotaran/analysis/__init__.py

```python
from glotaran.deprecation.deprecation_utils import deprecate_submodule

result = deprecate_submodule(
    deprecated_module_name="glotaran.analysis.result",
    new_module_name="glotaran.project.result",
    to_be_removed_in_version="0.6.0",
)
```

glotaran_version

`glotaran.deprecation.deprecation_utils.`**`glotaran_version`**`()` → str

Version of the distribution.

This is basically the same as `glotaran.__version__` but independent from glotaran. This way all of the deprecation functionality can be used even in `glotaran.__init__.py` without moving the import below the definition of `__version__` or causeing a circular import issue.

Returns The version string.

Return type str

module_attribute

`glotaran.deprecation.deprecation_utils.`**`module_attribute`**`(`*module_qual_name: str*, *attribute_name: str*`)` → Any

Import and return the attribute (e.g. function or class) of a module.

This is basically the same as `from module_name import attribute_name as return_value` where this function returns `return_value`.

Parameters

- **`module_qual_name`** (*str*) – Fully qualified name for a module e.g. `glotaran.model.base_model`
- **`attribute_name`** (*str*) – Name of the attribute e.g. `Model`

Returns Attribute of the module, e.g. a function or class.

Return type Any

parse_version

glotaran.deprecation.deprecation_utils.**parse_version**(*version_str: str*) → tuple[int, int, int]

Parse version string to tuple of three ints for comparison.

Parameters **version_str** (*str*) – Fully qualified version string of the form 'major.minor.patch'.

Returns Version as tuple.

Return type tuple[int, int, int]

Raises

- **ValueError** – If `version_str` has less that three elements separated by `.`.
- **ValueError** – If `version_str` 's first three elements can not be casted to int.

warn_deprecated

glotaran.deprecation.deprecation_utils.**warn_deprecated**(*, *deprecated_qual_name_usage: str, new_qual_name_usage: str, to_be_removed_in_version: str, check_qual_names: tuple[bool, bool] = (True, True), stacklevel: int = 2, importable_indices: tuple[int, int] = (1, 1)*) → None

Raise deprecation warning with change information.

The change information are old / new usage information and end of support version.

Parameters

- **deprecated_qual_name_usage** (*str*) – Old usage with fully qualified name e.g.: `'glotaran.read_model_from_yaml(model_yml_str)'`
- **new_qual_name_usage** (*str*) – New usage as fully qualified name e.g.: `'glotaran.io.load_model(model_yml_str, format_name="yml_str")'`
- **to_be_removed_in_version** (*str*) – Version the support for this usage will be removed.
- **check_qual_names** (*tuple[bool, bool]*) – Whether or not to check for the existence `deprecated_qual_name_usage` and `deprecated_qual_name_usage`
  - Set the first value to False to prevent infinite recursion error when changing a module attribute import.
  - Set the second value to False if the new usage in in a different package or there is none.
- **stacklevel** (*int*) – Stack at which the warning should be shown as raise. Default: 2
- **importable_indices** (*tuple[int, int]*) – Indices from right for most nested item which is importable for `deprecated_qual_name_usage` and `new_qual_name_usage` after splitting at `.`. This is used when

the old or new usage is a method or mapping access. E.g. let `deprecated_qual_name_usage` be `package.module.class.mapping["key"]`, then you would use `importable_indices=(2, 1)`, this way func:*check_qualnames_in_tests* will import `package.module.class` and check if `class` has an attribute `mapping`.

Warns OverDueDeprecation – If the current version is greater or equal to `end_of_life_version`.

See also:

deprecate, *deprecate_module_attribute*, *deprecate_submodule*, *check_qualnames_in_tests*

Examples

This is the way the old `read_parameters_from_yaml_file` could deprecated and the usage of `load_model` being promoted instead.

Listing 4: glotaran/deprecation/modules/glotaran_root.py

```
def read_parameters_from_yaml_file(model_path: str):
    warn_deprecated(
        deprecated_qual_name_usage="glotaran.read_parameters_from_yaml_
↪file(model_path)",
        new_qual_name_usage="glotaran.io.load_model.load_model(model_path)
↪",
        to_be_removed_in_version="0.6.0",
    )
    return load_model(model_path)
```

Exceptions

Exception Summary

OverDueDeprecation	Error thrown when a deprecation should have been removed.

OverDueDeprecation

exception glotaran.deprecation.deprecation_utils.**OverDueDeprecation**

Error thrown when a deprecation should have been removed.

See also:

deprecate, *warn_deprecated*, *deprecate_module_attribute*, *deprecate_submodule*

modules

Package containing deprecated implementations which were removed.

To keep things organized the filenames should be like the relative import path from glotaran root, but with `_` instead of `.`. E.g. `glotaran.analysis.scheme` would map to `analysis_scheme.py`

The only exceptions to this rule are the root `__init__.py` which is named `glotaran_root.py` and testing changed imports which should be placed in `test_changed_imports.py`.

Modules

glotaran.deprecation.modules.glotaran_root	Deprecated attributes from `glotaran.__init__` which are removed.

glotaran_root

Deprecated attributes from `glotaran.__init__` which are removed.

Functions

Summary

read_model_from_yaml	Parse `yaml` string to `Model`.
read_model_from_yaml_file	Parse `model.yaml` file to `Model`.
read_parameters_from_csv_file	Parse `parameters_file` to `ParameterGroup`.
read_parameters_from_yaml	Parse `yaml` string to `ParameterGroup`.
read_parameters_from_yaml_file	Parse `parameters_file` to `ParameterGroup`.

read_model_from_yaml

`glotaran.deprecation.modules.glotaran_root.`**`read_model_from_yaml`**(*model_yml_str: str*) → Model

Parse `yaml` string to `Model`.

> **Warning:** Deprecated use `glotaran.io.load_model(model_yml_str, format_name="yml_str")` instead.

Parameters **`model_yml_str`** (*str*) – Model spec description in yaml.

Returns Model described in `model_yml_str`.

Return type *Model*

read_model_from_yaml_file

`glotaran.deprecation.modules.glotaran_root.`**`read_model_from_yaml_file`**(*model_file: str*) → Model

Parse `model.yaml` file to `Model`.

> **Warning:** Deprecated use `glotaran.io.load_model(model_file)` instead.

Parameters **`model_file`** (*`str`*) – File with model spec description as yaml.

Returns Model described in `model_file`.

Return type *Model*

read_parameters_from_csv_file

`glotaran.deprecation.modules.glotaran_root.`**`read_parameters_from_csv_file`**(*parameters_file: str*) → ParameterGroup

Parse `parameters_file` to `ParameterGroup`.

> **Warning:** Deprecated use `glotaran.io.load_parameters(parameters_file)` instead.

Parameters **`parameters_file`** (*`str`*) – File with parameters in csv.

Returns ParameterGroup described in `parameters_file`.

Return type *ParameterGroup*

read_parameters_from_yaml

`glotaran.deprecation.modules.glotaran_root.`**`read_parameters_from_yaml`**(*parameters_yml_str: str*) → ParameterGroup

Parse `yaml` string to `ParameterGroup`.

> **Warning:** Deprecated use `glotaran.io.load_parameters(parameters_yml_str, format_name="yml_str")` instead.

Parameters **`parameters_yml_str`** (*`str`*) – PArameter spec description in yaml.

Returns ParameterGroup described in `parameters_yml_str`.

Return type *ParameterGroup*

read_parameters_from_yaml_file

glotaran.deprecation.modules.glotaran_root.**read_parameters_from_yaml_file**(*parameters_file: str*) → ParameterGroup

Parse `parameters_file` to `ParameterGroup`.

> **Warning:** Deprecated use `glotaran.io.load_parameters(parameters_file)` instead.

Parameters **parameters_file** (*str*) – File with parameters in yaml.

Returns ParameterGroup described in `parameters_file`.

Return type *ParameterGroup*

12.1.5 examples

Modules

glotaran.examples.sequential	

sequential

12.1.6 io

Functions for data IO

Note:

Since Io functionality is purely plugin based this package mostly reexports functions from the pluginsystem from a common place.

Modules

glotaran.io.interface	Baseclasses to create Data/Project IO plugins from.
glotaran.io.prepare_dataset	

interface

Baseclasses to create Data/Project IO plugins from.

The main purpose of those classes are to guarantee a consistent API via typechecker like `mypy` and demonstarate with methods are accessed by highlevel convenience functions for a given type of plugin.

To add additional options to a method, those options need to be keyword only arguments. See: https://www.python.org/dev/peps/pep-3102/

Classes

Summary

DataIoInterface	Baseclass for Data IO plugins.
ProjectIoInterface	Baseclass for Project IO plugins.

DataIoInterface

class `glotaran.io.interface.`**`DataIoInterface`**(*format_name: str*)

Bases: `object`

Baseclass for Data IO plugins.

Initialize a Data IO plugin with the name of the format.

Parameters **`format_name`** (*str*) – Name of the supported format an instance uses.

Methods Summary

load_dataset	Read data from a file to xarray.Dataset or xarray.DataArray (**NOT IMPLEMENTED**).
save_dataset	Save data from xarray.Dataset to a file (**NOT IMPLEMENTED**).

load_dataset

`DataIoInterface.`**`load_dataset`**(*file_name: str*) → xr.Dataset | xr.DataArray

Read data from a file to xarray.Dataset or xarray.DataArray (**NOT IMPLEMENTED**).

Parameters **`file_name`** (*str*) – File containing the data.

Returns Data loaded from the file.

Return type xr.Dataset|xr.DataArray

save_dataset

`DataIoInterface.`**`save_dataset`**(*dataset: xr.Dataset | xr.DataArray*, *file_name: str*)

Save data from xarray.Dataset to a file (**NOT IMPLEMENTED**).

Parameters

- **`dataset`** (*`xr.Dataset`*) – Dataset to be saved to file.
- **`file_name`** (*`str`*) – File to write the data to.

Methods Documentation

`load_dataset`(*file_name: str*) → xr.Dataset | xr.DataArray

Read data from a file to xarray.Dataset or xarray.DataArray (**NOT IMPLEMENTED**).

Parameters **`file_name`** (*`str`*) – File containing the data.

Returns Data loaded from the file.

Return type xr.Dataset|xr.DataArray

`save_dataset`(*dataset: xr.Dataset | xr.DataArray*, *file_name: str*)

Save data from xarray.Dataset to a file (**NOT IMPLEMENTED**).

Parameters

- **`dataset`** (*`xr.Dataset`*) – Dataset to be saved to file.
- **`file_name`** (*`str`*) – File to write the data to.

ProjectIoInterface

class `glotaran.io.interface.`**`ProjectIoInterface`**(*format_name: str*)

Bases: `object`

Baseclass for Project IO plugins.

Initialize a Project IO plugin with the name of the format.

Parameters **`format_name`** (*`str`*) – Name of the supported format an instance uses.

Methods Summary

`load_model`	Create a Model instance from the specs defined in a file (**NOT IMPLEMENTED**).
`load_parameters`	Create a ParameterGroup instance from the specs defined in a file (**NOT IMPLEMENTED**).
`load_result`	Create a Result instance from the specs defined in a file (**NOT IMPLEMENTED**).
`load_scheme`	Create a Scheme instance from the specs defined in a file (**NOT IMPLEMENTED**).
`save_model`	Save a Model instance to a spec file (**NOT IMPLEMENTED**).

continues on next page

Table 163 – continued from previous page

save_parameters	Save a ParameterGroup instance to a spec file (**NOT IMPLEMENTED**).
save_result	Save a Result instance to a spec file (**NOT IMPLEMENTED**).
save_scheme	Save a Scheme instance to a spec file (**NOT IMPLEMENTED**).

load_model

ProjectIoInterface.**load_model**(*file_name: str*) → Model

Create a Model instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **file_name** (*str*) – File containing the model specs.

Returns Model instance created from the file.

Return type *Model*

load_parameters

ProjectIoInterface.**load_parameters**(*file_name: str*) → ParameterGroup

Create a ParameterGroup instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **file_name** (*str*) – File containing the parameter specs.

Returns ParameterGroup instance created from the file.

Return type *ParameterGroup*

load_result

ProjectIoInterface.**load_result**(*result_path: str*) → Result

Create a Result instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **result_path** (*str*) – Path containing the result data.

Returns Result instance created from the file.

Return type *Result*

load_scheme

ProjectIoInterface.**load_scheme**(*file_name: str*) → Scheme

Create a Scheme instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **file_name** (*str*) – File containing the parameter specs.

Returns

- *Scheme* – Scheme instance created from the file.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

save_model

`ProjectIoInterface.`**`save_model`**`(`*model: Model*, *file_name: str*`)`

Save a Model instance to a spec file (**NOT IMPLEMENTED**).

Parameters

- **`model`** (`Model`) – Model instance to save to specs file.
- **`file_name`** (*`str`*) – File to write the model specs to.

save_parameters

`ProjectIoInterface.`**`save_parameters`**`(`*parameters: ParameterGroup*, *file_name: str*`)`

Save a ParameterGroup instance to a spec file (**NOT IMPLEMENTED**).

Parameters

- **`parameters`** (`ParameterGroup`) – ParameterGroup instance to save to specs file.
- **`file_name`** (*`str`*) – File to write the parameter specs to.

save_result

`ProjectIoInterface.`**`save_result`**`(`*result: Result*, *result_path: str*`)`

Save a Result instance to a spec file (**NOT IMPLEMENTED**).

Parameters

- **`result`** (`Result`) – Result instance to save to specs file.
- **`result_path`** (*`str`*) – Path to write the result data to.

save_scheme

`ProjectIoInterface.`**`save_scheme`**`(`*scheme: Scheme*, *file_name: str*`)`

Save a Scheme instance to a spec file (**NOT IMPLEMENTED**).

Parameters

- **`scheme`** (`Scheme`) – Scheme instance to save to specs file.
- **`file_name`** (*`str`*) – File to write the scheme specs to.

Methods Documentation

`load_model``(`*file_name: str*`)` → Model

Create a Model instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **`file_name`** (*`str`*) – File containing the model specs.

Returns Model instance created from the file.

Return type *Model*

`load_parameters``(`*file_name: str*`)` → ParameterGroup

Create a ParameterGroup instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **file_name** (*str*) – File containing the parameter specs.

Returns ParameterGroup instance created from the file.

Return type *ParameterGroup*

load_result(*result_path: str*) → Result
Create a Result instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **result_path** (*str*) – Path containing the result data.

Returns Result instance created from the file.

Return type *Result*

load_scheme(*file_name: str*) → Scheme
Create a Scheme instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **file_name** (*str*) – File containing the parameter specs.

Returns

- *Scheme* – Scheme instance created from the file.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

save_model(*model: Model*, *file_name: str*)
Save a Model instance to a spec file (**NOT IMPLEMENTED**).

Parameters

- **model** (Model) – Model instance to save to specs file.
- **file_name** (*str*) – File to write the model specs to.

save_parameters(*parameters: ParameterGroup*, *file_name: str*)
Save a ParameterGroup instance to a spec file (**NOT IMPLEMENTED**).

Parameters

- **parameters** (ParameterGroup) – ParameterGroup instance to save to specs file.
- **file_name** (*str*) – File to write the parameter specs to.

save_result(*result: Result*, *result_path: str*)
Save a Result instance to a spec file (**NOT IMPLEMENTED**).

Parameters

- **result** (Result) – Result instance to save to specs file.
- **result_path** (*str*) – Path to write the result data to.

save_scheme(*scheme: Scheme*, *file_name: str*)
Save a Scheme instance to a spec file (**NOT IMPLEMENTED**).

Parameters

- **scheme** (Scheme) – Scheme instance to save to specs file.
- **file_name** (*str*) – File to write the scheme specs to.

prepare_dataset

Functions

Summary

add_svd_to_dataset	Add the SVD of a dataset inplace as Data variables to the dataset.
prepare_time_trace_dataset	Prepares a time trace for global analysis.

add_svd_to_dataset

glotaran.io.prepare_dataset.**add_svd_to_dataset**(*dataset: xr.Dataset*, *name: str = 'data'*, *lsv_dim: Hashable = 'time'*, *rsv_dim: Hashable = 'spectral'*, *data_array: xr.DataArray = None*)

Add the SVD of a dataset inplace as Data variables to the dataset.

The SVD is only computed if it doesn't already exist on the dataset.

Parameters

- **dataset** (*xr.Dataset*) – Dataset the SVD values should be added to.
- **name** (*str*) – Key to access the datarray inside of the dataset, by default "data"
- **lsv_dim** (*Hashable*) – Name of the dimension for the left singular value, by default "time"
- **rsv_dim** (*Hashable*) – Name of the dimension for the right singular value, by default "spectral"
- **data_array** (*xr.DataArray*) – Dataarray to calculate the SVD for, when provided the data extraction from the dataset will be skipped, by default None

prepare_time_trace_dataset

glotaran.io.prepare_dataset.**prepare_time_trace_dataset**(*dataset: xr.DataArray | xr.Dataset*, *weight: np.ndarray = None*, *irf: np.ndarray | xr.DataArray = None*) → xr.Dataset

Prepares a time trace for global analysis.

Parameters

- **dataset** – The dataset.
- **weight** – A weight for the dataset.
- **irf** – An IRF for the dataset.

12.1.7 model

Glotaran Model Package

This package contains the Glotaran's base model object, the model decorators and common model items.

Modules

glotaran.model.attribute	The model attribute decorator.
glotaran.model.base_model	A base class for global analysis models.
glotaran.model.dataset_descriptor	The DatasetDescriptor class.
glotaran.model.decorator	The model decorator.
glotaran.model.megacomplex	
glotaran.model.property	The model property class.
glotaran.model.util	Helper functions.
glotaran.model.weight	The Weight property class.

attribute

The model attribute decorator.

Functions

Summary

model_attribute	The *@model_attribute* decorator adds the given properties to the class.
model_attribute_typed	The model_attribute_typed decorator adds attributes to the class to enable the glotaran model parser to infer the correct class for an item when there are multiple variants.

model_attribute

glotaran.model.attribute.**model_attribute**(*properties: Any | dict[str, dict[str, Any]] = {}*, *has_type: bool = False*, *no_label: bool = False*) → Callable

The *@model_attribute* decorator adds the given properties to the class. Further it adds classmethods for deserialization, validation and printing.

By default, a *label* property is added.

The *properties* dictionary contains the name of the properties as keys. The values must be either a *type* or dictionary with the following values:

- type: a *type* (required)
- doc: a string for documentation (optional)
- default: a default value (optional)

- allow_none: if *True*, the property can be set to None (optional)

Classes with the *model_attribute* decorator intended to be used in glotaran models.

Parameters

- **properties** – A dictionary of property names and options.
- **has_type** – If true, a type property will added. Used for model attributes, which can have more then one type.
- **no_label** – If true no label property will be added.

model_attribute_typed

`glotaran.model.attribute.model_attribute_typed(types: dict[str, Any], no_label=False, default_type: str = None)`

The model_attribute_typed decorator adds attributes to the class to enable the glotaran model parser to infer the correct class for an item when there are multiple variants.

Parameters

- **types** – A dictionary of types and options.
- **no_label** – If *True* no label property will be added.

base_model

A base class for global analysis models.

Classes

Summary

Model	A base class for global analysis models.

Model

`class glotaran.model.base_model.Model`

Bases: `object`

A base class for global analysis models.

Attributes Summary

model_type	The type of the model as human readable string.

model_type

Model.**model_type**
The type of the model as human readable string.

Methods Summary

from_dict	Creates a model from a dictionary.
markdown	Formats the model as Markdown string.
problem_list	Returns a list with all problems in the model and missing parameters if specified.
simulate	Simulates the model.
valid	Returns *True* if the number problems in the model is 0, else *False*
validate	Returns a string listing all problems in the model and missing parameters if specified.

from_dict

classmethod Model.**from_dict**(*model_dict_ref: dict*) → *glotaran.model.base_model.Model*
Creates a model from a dictionary.

Parameters **model_dict** – Dictionary containing the model.

markdown

Model.**markdown**(*parameters: Optional[*glotaran.parameter.parameter_group.ParameterGroup*] = None*, *initial_parameters: Optional[*glotaran.parameter.parameter_group.ParameterGroup*] = None*, *base_heading_level: int = 1*) → *glotaran.utils.ipython.MarkdownStr*
Formats the model as Markdown string.

Parameters will be included if specified.

Parameters

- **parameter** (ParameterGroup) – Parameter to include.
- **initial_parameters** (ParameterGroup) – Initial values for the parameters.
- **base_heading_level** (*int*) – Base heading level of the markdown sections.

  E.g.:

  – If it is 1 the string will start with '# Model'.

  – If it is 3 the string will start with '### Model'.

problem_list

`Model.`**`problem_list`**(*parameters: ParameterGroup = None*) → list[str]

Returns a list with all problems in the model and missing parameters if specified.

Parameters **`parameter`** – The parameter to validate.

simulate

`Model.`**`simulate`**(*dataset: str*, *parameters: ParameterGroup*, *axes: dict[str, np.ndarray] = None*, *clp: np.ndarray | xr.DataArray = None*, *noise: bool = False*, *noise_std_dev: float = 1.0*, *noise_seed: int = None*) → xr.Dataset

Simulates the model.

Parameters

- **`dataset`** – Label of the dataset to simulate.
- **`parameter`** – The parameters for the simulation.
- **`axes`** – A dictionary with axes for simulation.
- **`clp`** – Conditionally linear parameters. Used instead of *model.global_matrix* if provided.
- **`noise`** – If *True* noise is added to the simulated data.
- **`noise_std_dev`** – The standard deviation of the noise.
- **`noise_seed`** – Seed for the noise.

valid

`Model.`**`valid`**(*parameters: Optional[*glotaran.parameter.parameter_group.ParameterGroup*] = None*) → bool

Returns *True* if the number problems in the model is 0, else *False*

Parameters **`parameter`** – The parameter to validate.

validate

`Model.`**`validate`**(*parameters: Optional[*glotaran.parameter.parameter_group.ParameterGroup*] = None*) → str

Returns a string listing all problems in the model and missing parameters if specified.

Parameters **`parameter`** – The parameter to validate.

Methods Documentation

classmethod from_dict(*model_dict_ref: dict*) → *glotaran.model.base_model.Model*

Creates a model from a dictionary.

Parameters `model_dict` – Dictionary containing the model.

markdown(*parameters: Optional[glotaran.parameter.parameter_group.ParameterGroup] = None, initial_parameters: Optional[glotaran.parameter.parameter_group.ParameterGroup] = None, base_heading_level: int = 1*) → *glotaran.utils.ipython.MarkdownStr*

Formats the model as Markdown string.

Parameters will be included if specified.

Parameters

- **parameter** (`ParameterGroup`) – Parameter to include.
- **initial_parameters** (`ParameterGroup`) – Initial values for the parameters.
- **base_heading_level** (*int*) – Base heading level of the markdown sections.

  E.g.:

  – If it is 1 the string will start with '# Model'.

  – If it is 3 the string will start with '### Model'.

property model_type: str

The type of the model as human readable string.

problem_list(*parameters: ParameterGroup = None*) → list[str]

Returns a list with all problems in the model and missing parameters if specified.

Parameters `parameter` – The parameter to validate.

simulate(*dataset: str, parameters: ParameterGroup, axes: dict[str, np.ndarray] = None, clp: np.ndarray | xr.DataArray = None, noise: bool = False, noise_std_dev: float = 1.0, noise_seed: int = None*) → xr.Dataset

Simulates the model.

Parameters

- **dataset** – Label of the dataset to simulate.
- **parameter** – The parameters for the simulation.
- **axes** – A dictionary with axes for simulation.
- **clp** – Conditionally linear parameters. Used instead of *model.global_matrix* if provided.
- **noise** – If *True* noise is added to the simulated data.
- **noise_std_dev** – The standard deviation of the noise.
- **noise_seed** – Seed for the noise.

valid(*parameters: Optional[glotaran.parameter.parameter_group.ParameterGroup] = None*) → bool

Returns *True* if the number problems in the model is 0, else *False*

Parameters `parameter` – The parameter to validate.

validate(*parameters: Optional[glotaran.parameter.parameter_group.ParameterGroup] = None*) → str

Returns a string listing all problems in the model and missing parameters if specified.

Parameters **parameter** – The parameter to validate.

dataset_descriptor

The DatasetDescriptor class.

Classes

Summary

DatasetDescriptor	A *DatasetDescriptor* describes a dataset in terms of a glotaran model.

DatasetDescriptor

class glotaran.model.dataset_descriptor.**DatasetDescriptor**

Bases: object

A *DatasetDescriptor* describes a dataset in terms of a glotaran model. It contains references to model items which describe the physical model for a given dataset.

A general dataset describtor assigns one or more megacomplexes and a scale parameter.

Attributes Summary

label
megacomplex
megacomplex_scale
scale

label

`DatasetDescriptor.label`

megacomplex

`DatasetDescriptor.megacomplex`

megacomplex_scale

`DatasetDescriptor.megacomplex_scale`

scale

`DatasetDescriptor.scale`

Methods Summary

fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	
iterate_megacomplexes	
mprint	
validate	

fill

`DatasetDescriptor.fill`(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

classmethod DatasetDescriptor.**from_dict**(*values: dict*) → cls

from_list

classmethod DatasetDescriptor.**from_list**(*values: list*) → cls

iterate_megacomplexes

DatasetDescriptor.**iterate_megacomplexes**() → Generator[tuple[Parameter | int, Megacomplex | str]]

mprint

DatasetDescriptor.**mprint**(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

validate

DatasetDescriptor.**validate**(*model: Model*, *parameters=None*) → list[str]

Methods Documentation

fill(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

iterate_megacomplexes() → Generator[tuple[Parameter | int, Megacomplex | str]]

property label: str

property megacomplex: List[str]

property megacomplex_scale: List[glotaran.parameter.parameter.Parameter]

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property scale: glotaran.parameter.parameter.Parameter

validate(*model: Model*, *parameters=None*) → list[str]

decorator

The model decorator.

Functions

Summary

`model`	The *@model* decorator is intended to be used on subclasses of `glotaran.model.Model`.

model

`glotaran.model.decorator.`**model**(*model_type: str, attributes: dict[str, Any] = None, dataset_type: type[DatasetDescriptor] = <class 'glotaran.model.dataset_descriptor.DatasetDescriptor'>, default_megacomplex_type: str = None, megacomplex_types: dict[str, Megacomplex] | type[Megacomplex] = None, global_matrix: GlobalMatrixFunction = None, model_dimension: str = None, global_dimension: str = None, has_matrix_constraints_function: Callable[[type[Model]], bool] = None, constrain_matrix_function: ConstrainMatrixFunction = None, retrieve_clp_function: RetrieveClpFunction = None, has_additional_penalty_function: Callable[[type[Model]], bool] = None, additional_penalty_function: PenaltyFunction = None, finalize_data_function: FinalizeFunction = None, grouped: bool | Callable[[type[Model]], bool] = False, index_dependent: bool | Callable[[type[Model]], bool] = False*) → Callable[[type[Model]], type[Model]]

The *@model* decorator is intended to be used on subclasses of `glotaran.model.Model`. It creates properties for the given attributes as well as functions to add access them. Also it adds the functions (e.g. for *matrix*) to the model ensures they are added wrapped in a correct way.

Parameters

- **model_type** (*str*) – Human readable string used by the parser to identify the correct model.
- **attributes** (*Dict[str, Any], optional*) – A dictionary of attribute names and types. All types must be decorated with the `glotaran.model.model_attribute()` decorator, by default None.
- **dataset_type** (*Type[DatasetDescriptor], optional*) – A subclass of

DatasetDescriptor, by default DatasetDescriptor

- **megacomplex_type** (*Any, optional*) – A class for the model megacomplexes. The class must be decorated with the glotaran.model.model_attribute() decorator, by default None
- **matrix** (*Union[MatrixFunction, IndexDependentMatrixFunction], optional*) – A function to calculate the matrix for the model, by default None
- **global_matrix** (*GlobalMatrixFunction, optional*) – A function to calculate the global matrix for the model, by default None
- **model_dimension** (*str, optional*) – The name of model matrix row dimension, by default None
- **global_dimension** (*str, optional*) – The name of model global matrix row dimension, by default None
- **has_matrix_constraints_function** (*Callable[[Type[Model]], bool], optional*) – True if the model as a constrain_matrix_function set, by default None
- **constrain_matrix_function** (*ConstrainMatrixFunction, optional*) – A function to constrain the global matrix for the model, by default None
- **retrieve_clp_function** (*RetrieveClpFunction, optional*) – A function to retrieve the full clp from the reduced, by default None
- **has_additional_penalty_function** (*Callable[[Type[Model]], bool], optional*) – True if model has a additional_penalty_function set, by default None
- **additional_penalty_function** (*PenaltyFunction, optional*) – A function to calculate additional penalties when optimizing the model, by default None
- **finalize_data_function** (*FinalizeFunction, optional*) – A function to finalize data after optimization, by default None
- **grouped** (*Union[bool, Callable[[Type[Model]], bool]], optional*) – True if model described a grouped problem, by default False
- **index_dependent** (*Union[bool, Callable[[Type[Model]], bool]], optional*) – True if model described a index dependent problem, by default False

Returns Returns a decorated model function

Return type Callable

Raises

- **ValueError** – If model implements meth:*has_matrix_constraints_function* but not meth:*constrain_matrix_function* and meth:*retrieve_clp_function*
- **ValueError** – If model implements meth:*has_additional_penalty_function* but not meth:*additional_penalty_function*

megacomplex

Classes

Summary

Megacomplex	

Megacomplex

`class glotaran.model.megacomplex.Megacomplex`

Bases: `object`

Attributes Summary

label	
type	

label

`Megacomplex.label`

type

`Megacomplex.type`

Methods Summary

calculate_matrix	
fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	
from_list	
mprint	
validate	

calculate_matrix

Megacomplex.**calculate_matrix**(*model*, *dataset_descriptor: DatasetDescriptor*, *indices: dict[str, int]*, *axis: dict[str, np.ndarray]*, ***kwargs*)

fill

Megacomplex.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

classmethod Megacomplex.**from_dict**(*values: dict*) → cls

from_list

classmethod Megacomplex.**from_list**(*values: list*) → cls

mprint

Megacomplex.**mprint**(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

validate

Megacomplex.**validate**(*model: Model*, *parameters=None*) → list[str]

Methods Documentation

calculate_matrix(*model*, *dataset_descriptor: DatasetDescriptor*, *indices: dict[str, int]*, *axis: dict[str, np.ndarray]*, ***kwargs*)

fill(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

property label: str

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

property type: str

validate(*model: Model*, *parameters=None*) → list[str]

property

The model property class.

Classes

Summary

ModelProperty

ModelProperty

class glotaran.model.property.**ModelProperty**(*cls*, *name*, *prop_type*, *doc*, *default*, *allow_none*)

Bases: property

Attributes Summary

allow_none
fdel
fget
fset

allow_none

`ModelProperty.allow_none`

fdel

`ModelProperty.fdel`

fget

`ModelProperty.fget`

fset

`ModelProperty.fset`

Methods Summary

deleter	Descriptor to change the deleter on a property.
fill	
getter	Descriptor to change the getter on a property.
setter	Descriptor to change the setter on a property.
validate	

deleter

`ModelProperty.deleter()`

Descriptor to change the deleter on a property.

fill

`ModelProperty.fill(`*value*, *model*, *parameter*`)`

getter

`ModelProperty.getter()`

Descriptor to change the getter on a property.

setter

ModelProperty.**setter**()

Descriptor to change the setter on a property.

validate

ModelProperty.**validate**(*value*, *model*, *parameters=None*) → List[str]

Methods Documentation

property allow_none: bool

deleter()

Descriptor to change the deleter on a property.

fdel

fget

fill(*value*, *model*, *parameter*)

fset

getter()

Descriptor to change the getter on a property.

setter()

Descriptor to change the setter on a property.

validate(*value*, *model*, *parameters=None*) → List[str]

util

Helper functions.

Functions

Summary

wrap_func_as_method	A decorator to wrap a function as class method.

wrap_func_as_method

glotaran.model.util.**wrap_func_as_method**(*cls: Any*, *name: str = None*, *annotations: dict[str, type] = None*, *doc: str = None*) → Callable[[DecoratedFunc], DecoratedFunc]

A decorator to wrap a function as class method.

Notes

Only for internal use.

Parameters

- **cls** – The class in which the function will be wrapped.
- **name** – The name of method. If *None*, the original function's name is used.
- **annotations** – The annotations of the method. If *None*, the original function's annotations are used.
- **doc** – The documentation of the method. If *None*, the original function's documentation is used.

Exceptions

Exception Summary

ModelError	Raised when a model contains errors.

ModelError

exception glotaran.model.util.**ModelError**(*error: str*)

Raised when a model contains errors.

weight

The Weight property class.

Classes

Summary

Weight	The *Weight* class describes a value by which a dataset will scaled.

Weight

class glotaran.model.weight.**Weight**

Bases: object

The *Weight* class describes a value by which a dataset will scaled.

global_interval and *model_interval* are optional. The whole range of the dataset will be used if not set.

Attributes Summary

datasets
global_interval
model_interval
value

datasets

Weight.**datasets**

global_interval

Weight.**global_interval**

model_interval

Weight.**model_interval**

value

Weight.**value**

Methods Summary

fill	Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.
from_dict	

continues on next page

Table 184 – continued from previous page

from_list	
mprint	
validate	

fill

Weight.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

from_dict

classmethod Weight.**from_dict**(*values: dict*) → cls

from_list

classmethod Weight.**from_list**(*values: list*) → cls

mprint

Weight.**mprint**(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

validate

Weight.**validate**(*model: Model*, *parameters=None*) → list[str]

Methods Documentation

property datasets: None

fill(*model: Model*, *parameters: ParameterGroup*) → cls

Returns a copy of the {cls._name} instance with all members which are Parameters are replaced by the value of the corresponding parameter in the parameter group.

Parameters

- **model** – A glotaran model.
- **parameter** (ParameterGroup) – The parameter group to fill from.

classmethod from_dict(*values: dict*) → cls

classmethod from_list(*values: list*) → cls

property global_interval: List[Tuple[float, float]]

property model_interval: List[Tuple[float, float]]

mprint(*parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → str

validate(*model: Model*, *parameters=None*) → list[str]

property value: float

12.1.8 parameter

Modules

glotaran.parameter.parameter	The parameter class.
glotaran.parameter.parameter_group	The parameter group class

parameter

The parameter class.

Classes

Summary

Keys	Keys for parameter options.
Parameter	A parameter for optimization.

Keys

class glotaran.parameter.parameter.**Keys**

Bases: `object`

Keys for parameter options.

Attributes Summary

EXPR
MAX
MIN
NON_NEG
VARY

EXPR

Keys.**EXPR = 'expr'**

MAX

Keys.**MAX = 'max'**

MIN

Keys.**MIN = 'min'**

NON_NEG

Keys.**NON_NEG = 'non-negative'**

VARY

Keys.**VARY = 'vary'**

Methods Summary

—

Methods Documentation

EXPR = 'expr'

MAX = 'max'

MIN = 'min'

NON_NEG = 'non-negative'

VARY = 'vary'

Parameter

class glotaran.parameter.parameter.**Parameter**(*label: str = None*, *full_label: str = None*, *expression: str = None*, *maximum: int | float = inf*, *minimum: int | float = - inf*, *non_negative: bool = False*, *value: float | int = nan*, *vary: bool = True*)

Bases: Protocol[numpy.typing._array_like._DType_co]

A parameter for optimization.

Optimization Parameter supporting numpy array operations.

Parameters

- **label** (*str, optional*) – The label of the parameter., by default None
- **full_label** (*str, optional*) – The label of the parameter with its path in a parameter group prepended. , by default None
- **expression** (*str, optional*) – Expression to calculate the parameters value from, e.g. if used in relation to another parameter. , by default None
- **maximum** (*int, optional*) – Upper boundary for the parameter to be varied to., by default np.inf
- **minimum** (*int, optional*) – Lower boundary for the parameter to be varied to., by default -np.inf
- **non_negative** (*bool, optional*) – Whether the parameter should always be bigger than zero., by default False
- **value** (*float, optional*) – Value of the parameter, by default np.nan
- **vary** (*bool, optional*) – Whether the parameter should be changed during optimization or not. , by default True

Attributes Summary

expression	Expression to calculate the parameters value from.
full_label	The label of the parameter with its path in a parameter group prepended.
label	Label of the parameter
maximum	The upper bound of the parameter.
minimum	The lower bound of the parameter.
non_negative	Indicates if the parameter is non-negativ.
standard_error	The standard error of the optimized parameter.
transformed_expression	The expression of the parameter transformed for evaluation within a *ParameterGroup*.
value	The value of the parameter
vary	Indicates if the parameter should be optimized.

expression

Parameter.**expression**

Expression to calculate the parameters value from.

This can used to set a relation to another parameter.

full_label

Parameter.**full_label**

The label of the parameter with its path in a parameter group prepended.

label

Parameter.**label**

Label of the parameter

maximum

Parameter.**maximum**

The upper bound of the parameter.

minimum

Parameter.**minimum**

The lower bound of the parameter.

non_negative

`Parameter.non_negative`

Indicates if the parameter is non-negativ.

If true, the parameter will be transformed with $p' = \log p$ and $p = \exp p'$.

Always *False* if *expression* is not *None*.

standard_error

`Parameter.standard_error`

The standard error of the optimized parameter.

transformed_expression

`Parameter.transformed_expression`

The expression of the parameter transformed for evaluation within a *ParameterGroup*.

value

`Parameter.value`

The value of the parameter

vary

`Parameter.vary`

Indicates if the parameter should be optimized.

Always *False* if *expression* is not *None*.

Methods Summary

from_list_or_value	Creates a parameter from a list or numeric value.
get_value_and_bounds_for_optimization	Gets the parameter value and bounds with expression and non-negative constraints applied.
set_from_group	Sets all values of the parameter to the values of the corresponding parameter in the group.
set_value_from_optimization	Sets the value from an optimization result and reverses non-negative transformation.
valid_label	Returns true if the *label* is valid string.

from_list_or_value

classmethod Parameter.**from_list_or_value**(*value: int | float | list*, *default_options: dict = None*, *label: str = None*) → Parameter

Creates a parameter from a list or numeric value.

Parameters

- **value** – The list or numeric value.
- **default_options** – A dictionary of default options.
- **label** – The label of the parameter.

get_value_and_bounds_for_optimization

Parameter.**get_value_and_bounds_for_optimization**() → tuple[float, float, float]

Gets the parameter value and bounds with expression and non-negative constraints applied.

set_from_group

Parameter.**set_from_group**(*group: ParameterGroup*)

Sets all values of the parameter to the values of the corresponding parameter in the group.

Notes

For internal use.

Parameters **group** – The `glotaran.parameter.ParameterGroup`.

set_value_from_optimization

Parameter.**set_value_from_optimization**(*value: float*)

Sets the value from an optimization result and reverses non-negative transformation.

valid_label

classmethod Parameter.**valid_label**(*label: str*) → bool

Returns true if the *label* is valid string.

Methods Documentation

property expression: str | None

Expression to calculate the parameters value from.

This can used to set a relation to another parameter.

classmethod from_list_or_value(*value: int | float | list*, *default_options: dict = None*, *label: str = None*) → *Parameter*

Creates a parameter from a list or numeric value.

Parameters

- **value** – The list or numeric value.
- **default_options** – A dictionary of default options.
- **label** – The label of the parameter.

property full_label: str
The label of the parameter with its path in a parameter group prepended.

get_value_and_bounds_for_optimization() → tuple[float, float, float]
Gets the parameter value and bounds with expression and non-negative constraints applied.

property label: str | None
Label of the parameter

property maximum: float
The upper bound of the parameter.

property minimum: float
The lower bound of the parameter.

property non_negative: bool
Indicates if the parameter is non-negativ.

If true, the parameter will be transformed with $p' = \log p$ and $p = \exp p'$.

Always *False* if *expression* is not *None*.

set_from_group(*group: ParameterGroup*)
Sets all values of the parameter to the values of the corresponding parameter in the group.

Notes

For internal use.

Parameters **group** – The `glotaran.parameter.ParameterGroup`.

set_value_from_optimization(*value: float*)
Sets the value from an optimization result and reverses non-negative transformation.

property standard_error: float
The standard error of the optimized parameter.

property transformed_expression: str | None
The expression of the parameter transformed for evaluation within a *ParameterGroup*.

classmethod valid_label(*label: str*) → bool
Returns true if the *label* is valid string.

property value: float
The value of the parameter

property vary: bool
Indicates if the parameter should be optimized.

Always *False* if *expression* is not *None*.

parameter_group

The parameter group class

Classes

Summary

ParameterGroup	Represents are group of parameters.

ParameterGroup

class glotaran.parameter.parameter_group.**ParameterGroup**(*) -> new empty dictionary dict(mapping) -> new dictionary initialized from a mapping object's (key, value) pairs dict(iterable) -> new dictionary initialized as if via: d = {} for k, v in iterable: d[k] = v dict(**kwargs) -> new dictionary initialized with the name=value pairs in the keyword argument list. For example: dict(one=1, two=2)*)

Bases: dict

Represents are group of parameters. Can contain other groups, creating a tree-like hierarchy.

Parameters **label** – The label of the group.

Attributes Summary

label	Label of the group.
root_group	Root of the group.

label

ParameterGroup.**label**

Label of the group.

root_group

ParameterGroup.**root_group**
Root of the group.

Methods Summary

add_group	Adds a *ParameterGroup* to the group.
add_parameter	Adds a Parameter to the group.
all	Returns a generator over all parameter in the group and it's subgroups together with their labels.
clear	
copy	
from_dataframe	Creates a *ParameterGroup* from a pandas. DataFrame
from_dict	Creates a *ParameterGroup* from a dictionary.
from_list	Creates a *ParameterGroup* from a list.
fromkeys	Create a new dictionary with keys from iterable and values set to value.
get	Gets a Parameter by its label.
get_label_value_and_bounds_arrays	Returns a arrays of all parameter labels, values and bounds.
get_nr_roots	Returns the number of roots of the group.
groups	Returns a generator over all groups and their subgroups.
has	Checks if a parameter with the given label is in the group or in a subgroup.
items	
keys	
markdown	Formats the *ParameterGroup* as markdown string.
pop	If key is not found, d is returned if given, otherwise KeyError is raised
popitem	Remove and return a (key, value) pair as a 2-tuple.
set_from_label_and_value_arrays	Updates the parameter values from a list of labels and values.
setdefault	Insert key with a value of default if key is not in the dictionary.
to_csv	Writes a *ParameterGroup* to a CSV file.
to_dataframe	

continues on next page

Table 193 – continued from previous page

update	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
update_parameter_expression	Updates all parameters which have an expression.
values	

add_group

`ParameterGroup.add_group(group: glotaran.parameter.parameter_group.ParameterGroup)`

Adds a *ParameterGroup* to the group.

Parameters **group** – The group to add.

add_parameter

`ParameterGroup.add_parameter(parameter: Parameter | list[Parameter])`

Adds a `Parameter` to the group.

Parameters **parameter** – The parameter to add.

all

`ParameterGroup.all(root: str = None, separator: str = '.')` → Generator[tuple[str, Parameter], None, None]

Returns a generator over all parameter in the group and it's subgroups together with their labels.

Parameters

- **root** – The label of the root group
- **separator** – The separator for the parameter labels.

clear

`ParameterGroup.clear()` → None. Remove all items from D.

copy

ParameterGroup.**copy**() → a shallow copy of D

from_dataframe

classmethod ParameterGroup.**from_dataframe**(*df: pandas.core.frame.DataFrame*, *source: str = 'DataFrame'*) → *glotaran.parameter.parameter_group.ParameterGroup*

Creates a ParameterGroup from a pandas.DataFrame

from_dict

classmethod ParameterGroup.**from_dict**(*parameter_dict: dict[str, dict | list]*, *label: str = None*, *root_group: ParameterGroup = None*) → ParameterGroup

Creates a ParameterGroup from a dictionary.

Parameters

- **parameter_dict** – A parameter dictionary containing parameters.
- **label** – The label of root group.
- **root_group** – The root group

from_list

classmethod ParameterGroup.**from_list**(*parameter_list: list[float | list]*, *label: str = None*, *root_group: ParameterGroup = None*) → ParameterGroup

Creates a ParameterGroup from a list.

Parameters

- **parameter_list** – A parameter list containing parameters
- **label** – The label of the root group.
- **root_group** – The root group

fromkeys

ParameterGroup.**fromkeys**(*iterable*, *value=None*, */*)

Create a new dictionary with keys from iterable and values set to value.

get

ParameterGroup.**get**(*label: str*) → *glotaran.parameter.parameter.Parameter*

Gets a Parameter by its label.

Parameters **label** – The label of the parameter, with its path in a parameter group prepended.

get_label_value_and_bounds_arrays

ParameterGroup.**get_label_value_and_bounds_arrays**(*exclude_non_vary: bool = False*) → tuple[list[str], np.ndarray, np.ndarray, np.ndarray]

Returns a arrays of all parameter labels, values and bounds.

Parameters **exclude_non_vary** (*bool = False*) – If true, parameters with *vary=False* are excluded.

get_nr_roots

ParameterGroup.**get_nr_roots**() → int

Returns the number of roots of the group.

groups

ParameterGroup.**groups**() → Generator[*glotaran.parameter.parameter_group.ParameterGroup*, None, None]

Returns a generator over all groups and their subgroups.

has

ParameterGroup.**has**(*label: str*) → bool

Checks if a parameter with the given label is in the group or in a subgroup.

Parameters **label** – The label of the parameter, with its path in a parameter group prepended.

items

ParameterGroup.**items**() → a set-like object providing a view on D's items

keys

ParameterGroup.**keys**() → a set-like object providing a view on D's keys

markdown

ParameterGroup.**markdown**() → *glotaran.utils.ipython.MarkdownStr*

Formats the `ParameterGroup` as markdown string.

This is done by recursing the nested `ParameterGroup` tree.

pop

ParameterGroup.**pop**(*k*[, *d*]) → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise KeyError is raised

popitem

ParameterGroup.**popitem**(/)

Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

set_from_label_and_value_arrays

ParameterGroup.**set_from_label_and_value_arrays**(*labels: list[str], values: np.ndarray*)

Updates the parameter values from a list of labels and values.

setdefault

ParameterGroup.**setdefault**(*key*, *default=None*, */*)

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

to_csv

ParameterGroup.**to_csv**(*filename: str*, *delimiter: str = ','*)

Writes a `ParameterGroup` to a CSV file.

Parameters

- **filepath** – The path to the CSV file.
- **delimiter** (*str*) – The delimiter of the CSV file.

to_dataframe

`ParameterGroup.to_dataframe()` → pandas.core.frame.DataFrame

update

`ParameterGroup.update([E], **F)` → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

update_parameter_expression

`ParameterGroup.update_parameter_expression()`

Updates all parameters which have an expression.

values

`ParameterGroup.values()` → an object providing a view on D's values

Methods Documentation

add_group(*group: glotaran.parameter.parameter_group.ParameterGroup*)

Adds a *ParameterGroup* to the group.

Parameters **group** – The group to add.

add_parameter(*parameter: Parameter | list[Parameter]*)

Adds a `Parameter` to the group.

Parameters **parameter** – The parameter to add.

all(*root: str = None, separator: str = '.'*) → Generator[tuple[str, Parameter], None, None]

Returns a generator over all parameter in the group and it's subgroups together with their labels.

Parameters

- **root** – The label of the root group
- **separator** – The separator for the parameter labels.

clear() → None. Remove all items from D.

copy() → a shallow copy of D

classmethod from_dataframe(*df: pandas.core.frame.DataFrame, source: str = 'DataFrame'*) → *glotaran.parameter.parameter_group.ParameterGroup*

Creates a *ParameterGroup* from a `pandas.DataFrame`

classmethod from_dict(*parameter_dict: dict[str, dict | list], label: str = None, root_group: ParameterGroup = None*) → *ParameterGroup*

Creates a `ParameterGroup` from a dictionary.

Parameters

- **parameter_dict** – A parameter dictionary containing parameters.
- **label** – The label of root group.
- **root_group** – The root group

classmethod from_list(*parameter_list: list[float | list], label: str = None, root_group: ParameterGroup = None*) → *ParameterGroup*

Creates a `ParameterGroup` from a list.

Parameters

- **parameter_list** – A parameter list containing parameters
- **label** – The label of the root group.
- **root_group** – The root group

fromkeys(*iterable, value=None, /*)

Create a new dictionary with keys from iterable and values set to value.

get(*label: str*) → *glotaran.parameter.parameter.Parameter*

Gets a `Parameter` by its label.

Parameters **label** – The label of the parameter, with its path in a parameter group prepended.

get_label_value_and_bounds_arrays(*exclude_non_vary: bool = False*) → tuple[list[str], np.ndarray, np.ndarray, np.ndarray]

Returns a arrays of all parameter labels, values and bounds.

Parameters **exclude_non_vary** (*bool = False*) – If true, parameters with *vary=False* are excluded.

get_nr_roots() → int

Returns the number of roots of the group.

groups() → Generator[*glotaran.parameter.parameter_group.ParameterGroup*, None, None]

Returns a generator over all groups and their subgroups.

has(*label: str*) → bool

Checks if a parameter with the given label is in the group or in a subgroup.

Parameters **label** – The label of the parameter, with its path in a parameter group prepended.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

property label: str

Label of the group.

markdown() → *glotaran.utils.ipython.MarkdownStr*

Formats the `ParameterGroup` as markdown string.

This is done by recursing the nested `ParameterGroup` tree.

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

popitem(/)
Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

property root_group: glotaran.parameter.parameter_group.ParameterGroup
Root of the group.

set_from_label_and_value_arrays(*labels: list[str]*, *values: np.ndarray*)
Updates the parameter values from a list of labels and values.

setdefault(*key*, *default=None*, /)
Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

to_csv(*filename: str*, *delimiter: str = ','*)
Writes a `ParameterGroup` to a CSV file.

Parameters

- **filepath** – The path to the CSV file.
- **delimiter** (*str*) – The delimiter of the CSV file.

to_dataframe() → pandas.core.frame.DataFrame

update([*E*], ***F*) → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

update_parameter_expression()
Updates all parameters which have an expression.

values() → an object providing a view on D's values

Exceptions

Exception Summary

ParameterNotFoundException	Raised when a Parameter is not found in the Group.

ParameterNotFoundException

exception glotaran.parameter.parameter_group.**ParameterNotFoundException**(*path*, *label*)

Raised when a Parameter is not found in the Group.

12.1.9 plugin_system

Plugin system package containing all plugin related implementations.

Modules

glotaran.plugin_system.base_registry	Functionality to register, initialize and retrieve glotaran plugins.
glotaran.plugin_system.data_io_registration	Data Io registration convenience functions.
glotaran.plugin_system.io_plugin_utils	Utility functions for io plugin.
glotaran.plugin_system.model_registration	Model registration convenience functions.
glotaran.plugin_system.project_io_registration	Project Io registration convenience functions.

base_registry

Functionality to register, initialize and retrieve glotaran plugins.

Since this module is imported at the root `__init__.py` file all other glotaran imports should be used for typechecking only in the 'if TYPE_CHECKING' block. This is to prevent issues with circular imports.

Functions

Summary

add_instantiated_plugin_to_registry	Add instances of plugin_class to the given registry.
add_plugin_to_registry	Add a plugin with name `plugin_register_key` to the given registry.
full_plugin_name	Full name of a plugin instance/class similar to the `repr`.
get_method_from_plugin	Retrieve a method callabe from an class or instance plugin.
get_plugin_from_registry	Retrieve a plugin with name `plugin_register_key` is registered in a given registry.
is_registered_plugin	Check if a plugin with name `plugin_register_key` is registered in the given registry.
load_plugins	Initialize plugins registered under the entrypoint 'glotaran.plugins'.

continues on next page

Table 196 – continued from previous page

methods_differ_from_baseclass	Check if a plugins methods implementation differ from its baseclass.
methods_differ_from_baseclass_table	Create table of which plugins methods differ from their baseclass.
registered_plugins	Names of the plugins in the given registry.
set_plugin	Set a plugins short name to a specific plugin referred by its full name.
show_method_help	Show help on a method as if it was called directly on it.

add_instantiated_plugin_to_registry

glotaran.plugin_system.base_registry.**add_instantiated_plugin_to_registry**(*plugin_register_keys: str | list[str], plugin_class: type[_PluginInstantiableType], plugin_registry: MutableMapping[str, _PluginInstantiableType], plugin_set_func_name: str*) → None

Add instances of plugin_class to the given registry.

Parameters

- **plugin_register_keys** (*str | list[str]*) – Name/-s of the plugin under which it is registered.
- **plugin_class** (*type[_PluginInstantiableType]*) – Pluginclass which should be instantiated with `plugin_register_keys` and added to the registry.
- **plugin_registry** (*MutableMapping[str, _PluginInstantiableType]*) – Registry the plugin should be added to.
- **plugin_set_func_name** (*str*) – Name of the function used to pin a plugin.

See also:

`add_plugin_to_register`

add_plugin_to_registry

glotaran.plugin_system.base_registry.**add_plugin_to_registry**(*plugin_register_key: str*, *plugin: _PluginType*, *plugin_registry: MutableMapping[str, _PluginType]*, *plugin_set_func_name: str*, *instance_identifier: str = ''*) → None

Add a plugin with name `plugin_register_key` to the given registry.

In addition it also adds the plugin with it full import path name as key, which allows for a better reproducibility in case there are conflicting plugins.

Parameters

- **plugin_register_key** (*str*) – Name of the plugin under which it is registered.
- **plugin** (*_PluginType*) – Plugin to be added to the registry.
- **plugin_registry** (*MutableMapping[str, _PluginType]*) – Registry the plugin should be added to.
- **plugin_set_func_name** (*str*) – Name of the function used to pin a plugin.
- **instance_identifier** (*str*) – Used to differentiate between plugin instances (e.g. different format for IO plugins)

Raises `ValueError` – If `plugin_register_key` has the character '.' in it.

See also:

`add_instantiated_plugin_to_register`, *full_plugin_name*

full_plugin_name

glotaran.plugin_system.base_registry.**full_plugin_name**(*plugin: object | type[object]*) → str

Full name of a plugin instance/class similar to the `repr`.

Parameters **plugin** (*object | type[object]*) – plugin instance/class

Examples

```
>>> from glotaran.builtin.io.sdt.sdt_file_reader import SdtDataIo
>>> full_plugin_name(SdtDataIo)
"glotaran.builtin.io.sdt.sdt_file_reader.SdtDataIo"
>>> full_plugin_name(SdtDataIo("sdt"))
"glotaran.builtin.io.sdt.sdt_file_reader.SdtDataIo"
```

Returns Full name of the plugin.

Return type str

get_method_from_plugin

`glotaran.plugin_system.base_registry.`**`get_method_from_plugin`**(*plugin: object | type[object], method_name: str*) → Callable[..., Any]

Retrieve a method callabe from an class or instance plugin.

Parameters

- **plugin** (*object | type[object],*) – Plugin instance or class.
- **method_name** (*str*) – Method name, e.g. load_model.

Returns Method callable.

Return type Callable[…, Any]

Raises

- **ValueError** – If plugin has an attribute with that name but it isn't callable.
- **ValueError** – If plugin misses the attribute.

get_plugin_from_registry

`glotaran.plugin_system.base_registry.`**`get_plugin_from_registry`**(*plugin_register_key: str, plugin_registry: MutableMapping[str, _PluginType], not_found_error_message: str*) → _PluginType

Retrieve a plugin with name `plugin_register_key` is registered in a given registry.

Parameters

- **plugin_register_key** (*str*) – Name of the plugin under which it is registered.
- **plugin_registry** (*MutableMapping[str, _PluginType]*) – Registry to search in.
- **not_found_error_message** (*str*) – Error message to be shown if the plugin wasn't found.

Returns Plugin from the plugin Registry.

Return type _PluginType

Raises **ValueError** – If there was no plugin registered under the name `plugin_register_key`.

is_registered_plugin

glotaran.plugin_system.base_registry.**is_registered_plugin**(*plugin_register_key: str, plugin_registry: MutableMapping[str, _PluginType]*) → bool

Check if a plugin with name `plugin_register_key` is registered in the given registry.

Parameters

- **plugin_register_key** (*str*) – Name of the plugin under which it is registered.
- **plugin_registry** (*MutableMapping[str, _PluginType]*) – Registry to search in.

Returns Whether or not a plugin is in the registry.

Return type bool

load_plugins

glotaran.plugin_system.base_registry.**load_plugins**()

Initialize plugins registered under the entrypoint 'glotaran.plugins'.

For an entry_point to be considered a glotaran plugin it just needs to start with 'glotaran.plugins', which allows for an easy extendability.

Currently used builtin entrypoints are:

- `glotaran.plugins.data_io`
- `glotaran.plugins.model`
- `glotaran.plugins.project_io`

methods_differ_from_baseclass

glotaran.plugin_system.base_registry.**methods_differ_from_baseclass**(*method_names: str | Sequence[str], plugin: GenericPluginInstance | type[GenericPluginInstance], base_class: type[GenericPluginInstance]*) → list[bool]

Check if a plugins methods implementation differ from its baseclass.

Based on the assumption that `base_class` didn't implement the methods (e.g. `DataIoInterface` or `ProjectIoInterface`), this can be used to to create a 'supported methods' list.

Parameters

- **method_names** (*str | list[str]*) – Name|s of the method|s
- **plugin** (*GenericPluginInstance | type[GenericPluginInstance]*) – Plugin class or instance.

- **base_class** (*type[GenericPluginInstance]*) – Base class the plugin inherited from.

Returns List containing whether or not a plugins method differs from the baseclasses.

Return type list[bool]

methods_differ_from_baseclass_table

glotaran.plugin_system.base_registry.**methods_differ_from_baseclass_table**(*method_names: str | Sequence[str], plugin_registry_keys: str | Sequence[str], get_plugin_function: Callable[[str], GenericPluginInstance | type[GenericPluginInstance]], base_class: type[GenericPluginInstance], plugin_names: bool = False*) → list[list[str | bool]]

Create table of which plugins methods differ from their baseclass.

This uses the assumption that all plugins have the same `base_class`.

The main purpose of this function is to show the user which plugin implements which methods differently than its baseclass.

Based on the assumption that `base_class` didn't implement the methods (e.g. `DataIoInterface` or `ProjectIoInterface`), this can be used to to create a 'supported methods' table.

Parameters

- **method_names** (*str | list[str]*) – Name|s of the method|s.
- **plugin_registry_keys** (*str | list[str]*) – Keys the plugins are registered under (e.g. return value of the implementation of func:*registered_plugins*)
- **get_plugin_function** (*Callable[[str], GenericPluginInstance | type[GenericPluginInstance]]*) – Function to get plugin from plugin registry.
- **base_class** (*type[GenericPluginInstance]*) – Base class the plugin inherited from.
- **plugin_names** (*bool*) – Whether or not to add the names of the plugins to the lists.

Returns Table like structure with the first value of each row being the plugin_registry_key and the others whether or not a plugins method differs from the baseclasses.

Return type list[list[str | bool]]

See also:

methods_differ_from_baseclass

registered_plugins

glotaran.plugin_system.base_registry.**registered_plugins**(*plugin_registry: MutableMapping[str, _PluginType], full_names: bool = False*) → list[str]

Names of the plugins in the given registry.

Parameters

- **plugin_registry** (*MutableMapping[str, _PluginType]*) – Registry to search in.
- **full_names** (*bool*) – Whether to display the full names the plugins are registered under as well.

Returns List of plugin names in plugin_registry.

Return type list[str]

set_plugin

glotaran.plugin_system.base_registry.**set_plugin**(*plugin_register_key: str, full_plugin_name: str, plugin_registry: MutableMapping[str, _PluginType], plugin_register_key_name: str = 'format_name'*) → None

Set a plugins short name to a specific plugin referred by its full name.

This can be used to ensure that a specific plugin is used in case there are conflicting plugins installed.

Parameters

- **plugin_register_key** (*str*) – Name of the plugin under which it is registered.
- **full_plugin_name** (*str*) – Full name (import path) of the registered plugin.
- **plugin_registry** (*MutableMapping[str, _PluginType]*) – Registry the plugin should be set in to.
- **plugin_register_key_name** (*str*) – Name of the arg passed plugin_register_key in the function that implements set_plugin.

Raises

- **ValueError** – If plugin_register_key has the character '.' in it.
- **ValueError** – If there isn't a registered plugin with the key full_plugin_name.

See also:

add_plugin_to_registry, *full_plugin_name*

show_method_help

`glotaran.plugin_system.base_registry.`**`show_method_help`**(*plugin: object | type[object], method_name: str*) → None

Show help on a method as if it was called directly on it.

Parameters

- **plugin** (*object | type[object],*) – Plugin instance or class.
- **method_name** (*str*) – Method name, e.g. load_model.

Exceptions

Exception Summary

PluginOverwriteWarning	Warning used if a plugin tries to overwrite and existing plugin.

PluginOverwriteWarning

exception `glotaran.plugin_system.base_registry.`**`PluginOverwriteWarning`**(**args: Any, old_key: str, old_plugin: object | type[object], new_plugin: object | type[object], plugin_set_func_name: str*)

Warning used if a plugin tries to overwrite and existing plugin.

Use old and new plugin and keys to give verbose warning message.

Parameters

- **old_key** (*str*) – Old registry key.
- **old_plugin** (*object | type[object]*) – Old plugin ('registry[old_key]').
- **new_plugin** (*object | type[object]*) – New Plugin ('registry[new_key]').
- **plugin_set_func_name** (*str*) – Name of the function used to pin a plugin.
- ***args** (*Any*) – Additional args passed to the super constructor.

data_io_registration

Data Io registration convenience functions.

Note: The [call-arg] type error would be raised since the base methods doesn't have a `**kwargs` argument, but we rather ignore this error here, than adding `**kwargs` to the base method and causing an [override] type error in the plugins implementation.

Functions

Summary

data_io_plugin_table	Return registered data io plugins and which functions they support as markdown table.
get_data_io	Retrieve a data io plugin from the data_io registry.
get_dataloader	Retrieve implementation of the `read_dataset` functionality for the format 'format_name'.
get_datasaver	Retrieve implementation of the `save_dataset` functionality for the format 'format_name'.
is_known_data_format	Check if a data format is in the data_io registry.
known_data_formats	Names of the registered data io plugins.
load_dataset	Read data from a file to xarray.Dataset or xarray.DataArray.
register_data_io	Register data io plugins to one or more formats.
save_dataset	Save data from xarray.Dataset or xarray.DataArray to a file.
set_data_plugin	Set the plugin used for a specific data format.
show_data_io_method_help	Show help for the implementation of data io plugin methods.

data_io_plugin_table

glotaran.plugin_system.data_io_registration.**data_io_plugin_table**(*, *plugin_names: bool = False*, *full_names: bool = False*) → *glotaran.utils.ipython.MarkdownStr*

Return registered data io plugins and which functions they support as markdown table.

This is especially useful when you work with new plugins.

Parameters

- **plugin_names** (*bool*) – Whether or not to add the names of the plugins to the table.
- **full_names** (*bool*) – Whether to display the full names the plugins are registered under as well.

Returns Markdown table of data io plugins.

Return type str

get_data_io

glotaran.plugin_system.data_io_registration.**get_data_io**(*format_name: str*) → *glotaran.io.interface.DataIoInterface*

Retrieve a data io plugin from the data_io registry.

Parameters **format_name** (*str*) – Name of the data io plugin under which it is registered.

Returns Data io plugin instance.

Return type *DataIoInterface*

get_dataloader

glotaran.plugin_system.data_io_registration.**get_dataloader**(*format_name: str*) → DataLoader

Retrieve implementation of the `read_dataset` functionality for the format 'format_name'.

This allows to get the proper help and autocomplete for the function, which is especially valuable if the function provides additional options.

Parameters **format_name** (*str*) – Format the dataloader should be able to read.

Returns Function to load data of format `format_name` as xarray.Dataset or xarray.DataArray.

Return type DataLoader

get_datasaver

glotaran.plugin_system.data_io_registration.**get_datasaver**(*format_name: str*) → DataSaver

Retrieve implementation of the `save_dataset` functionality for the format 'format_name'.

This allows to get the proper help and autocomplete for the function, which is especially valuable if the function provides additional options.

Parameters **format_name** (*str*) – Format the datawriter should be able to write.

Returns Function to write xarray.Dataset to the format `format_name` .

Return type DataSaver

is_known_data_format

glotaran.plugin_system.data_io_registration.**is_known_data_format**(*format_name: str*) → bool

Check if a data format is in the data_io registry.

Parameters **format_name** (*str*) – Name of the data io plugin under which it is registered.

Returns Whether or not the data format is a registered data io plugins.

Return type bool

known_data_formats

`glotaran.plugin_system.data_io_registration.`**`known_data_formats`**`(`*full_names: bool = False*`)` → list[str]

Names of the registered data io plugins.

Parameters **`full_names`** (*`bool`*) – Whether to display the full names the plugins are registered under as well.

Returns List of registered data io plugins.

Return type list[str]

load_dataset

`glotaran.plugin_system.data_io_registration.`**`load_dataset`**`(`*file_name: str | PathLike[str], format_name: str = None, **kwargs: Any*`)` → xr.Dataset | xr.DataArray

Read data from a file to xarray.Dataset or xarray.DataArray.

Parameters

- **`file_name`** (*`str | PathLike[str]`*) – File containing the data.
- **`format_name`** (*`str`*) – Format the file is in, if not provided it will be inferred from the file extension.
- **`**kwargs`** (*`Any`*) – Additional keyword arguments passes to the `read_dataset` implementation of the data io plugin. If you aren't sure about those use `get_dataloader` to get the implementation with the proper help and autocomplete.

Returns Data loaded from the file.

Return type xr.Dataset|xr.DataArray

register_data_io

`glotaran.plugin_system.data_io_registration.`**`register_data_io`**`(`*format_names: str | list[str]*`)` → Callable[[type[DataIoInterface]], type[DataIoInterface]]

Register data io plugins to one or more formats.

Decorate a data io plugin class with `@register_data_io(format_name|[*format_names])` to add it to the registry.

Parameters **`format_names`** (*`str | list[str]`*) – Name of the data io plugin under which it is registered.

Returns Inner decorator function.

Return type Callable[[type[*DataIoInterface*]], type[*DataIoInterface*]]

Examples

```
>>> @register_data_io("my_format_1")
... class MyDataIo1(DataIoInterface):
...     pass
```

```
>>> @register_data_io(["my_format_1", "my_format_1_alias"])
... class MyDataIo2(DataIoInterface):
...     pass
```

save_dataset

glotaran.plugin_system.data_io_registration.**save_dataset**(*dataset: xr.Dataset | xr.DataArray*, *file_name: str | PathLike[str]*, *format_name: str = None*, ***, *allow_overwrite: bool = False*, ***kwargs: Any*) → None

Save data from xarray.Dataset or xarray.DataArray to a file.

Parameters

- **dataset** (*xr.Dataset | xr.DataArray*) – Data to be written to file.
- **file_name** (*str | PathLike[str]*) – File to write the data to.
- **format_name** (*str*) – Format the file should be in, if not provided it will be inferred from the file extension.
- **allow_overwrite** (*bool*) – Whether or not to allow overwriting existing files, by default False
- ****kwargs** (*Any*) – Additional keyword arguments passes to the `write_dataset` implementation of the data io plugin. If you aren't sure about those use `get_datawriter` to get the implementation with the proper help and autocomplete.

set_data_plugin

glotaran.plugin_system.data_io_registration.**set_data_plugin**(*format_name: str*, *full_plugin_name: str*) → None

Set the plugin used for a specific data format.

This function is useful when you want to resolve conflicts of installed plugins or overwrite the plugin used for a specific format.

Effected functions:

- *load_dataset()*
- *save_dataset()*

Parameters

- **format_name** (*str*) – Format name used to refer to the plugin when used for `save` and `load` functions.
- **full_plugin_name** (*str*) – Full name (import path) of the registered plugin.

show_data_io_method_help

`glotaran.plugin_system.data_io_registration.`**`show_data_io_method_help`**(*format_name: str, method_name: Literal[load_dataset, save_dataset]*) → None

Show help for the implementation of data io plugin methods.

Parameters

- **format_name** (*str*) – Format the method should support.
- **method_name** (*{'load_dataset', 'save_dataset'}*) – Method name

io_plugin_utils

Utility functions for io plugin.

Functions

Summary

bool_str_repr	Replace boolean value with string repr.
bool_table_repr	Replace boolean value with string repr for all table values.
inferr_file_format	Inferr format of a file if it exists.
not_implemented_to_value_error	Decorate a function to raise ValueError instead of NotImplementedError.
protect_from_overwrite	Raise FileExistsError if files already exists and allow_overwrite isn't True.

bool_str_repr

`glotaran.plugin_system.io_plugin_utils.`**`bool_str_repr`**(*value: Any, true_repr: str = '*', false_repr: str = '/'*) → Any

Replace boolean value with string repr.

This function is a helper for table representation (e.g. with tabulate) of boolean values.

Parameters

- **value** (*Any*) – Arbitrary value
- **true_repr** (*str*) – Desired repr for `True`, by default "*"

- **false_repr** (*str*) – Desired repr for `False`, by default "/"

Returns Original value or desired repr for bool

Return type Any

Examples

```
>>> table_data = [["foo", True, False], ["bar", False, True]]
>>> print(tabulate(map(lambda x: map(bool_table_repr, x), table_data)))
---  -  -
foo  *  /
bar  /  *
---  -  -
```

bool_table_repr

`glotaran.plugin_system.io_plugin_utils.bool_table_repr`(*table_data: Iterable[Iterable[Any]], true_repr: str = '*', false_repr: str = '/'*) → Iterator[Iterator[Any]]

Replace boolean value with string repr for all table values.

This function is an implementation of `bool_str_repr()` for a 2D table, for easy usage with tabulate.

Parameters

- **table_data** (*Iterable[Iterable[Any]]*) – Data of the table e.g. a list of lists.
- **true_repr** (*str*) – Desired repr for `True`, by default "*"
- **false_repr** (*str*) – Desired repr for `False`, by default "/"

Returns `table_data` with original values or desired repr for bool

Return type Iterator[Iterator[Any]]

See also:

bool_str_repr

Examples

```
>>> table_data = [["foo", True, False], ["bar", False, True]]
>>> print(tabulate(bool_table_repr(table_data))
---  -  -
foo  *  /
bar  /  *
---  -  -
```

inferr_file_format

glotaran.plugin_system.io_plugin_utils.**inferr_file_format**(*file_path: str | os.PathLike[str], *, needs_to_exist: bool = True, allow_folder=False*) → str

Inferr format of a file if it exists.

Parameters

- **file_path** (*str*) – Path/str to the file.
- **needs_to_exist** (*bool*) – Whether or not a file need to exists for an successful format inferring. While write functions don't need the file to exists, load functions do.
- **allow_folder** (*bool*) – Whether or not to allow the format to be `folder`. This is only used in `save_result`.

Returns File extension without the leading dot.

Return type str

Raises

- **ValueError** – If file doesn't exists.
- **ValueError** – If file has no extension.

not_implemented_to_value_error

glotaran.plugin_system.io_plugin_utils.**not_implemented_to_value_error**(*func: glotaran.plugin_system.io_plugin_uti*) → glotaran.plugin_system.io_plugin_uti

Decorate a function to raise ValueError instead of NotImplementedError.

This decorator is supposed to be used on functions which call functions that might raise a NotImplementedError, but raise ValueError instead with the same error text.

Parameters **func** (*DecoratedFunc*) – Function to be decorated.

Returns Wrapped function.

Return type DecoratedFunc

protect_from_overwrite

glotaran.plugin_system.io_plugin_utils.**protect_from_overwrite**(*path: str | os.PathLike[str], *, allow_overwrite: bool = False*) → None

Raise FileExistsError if files already exists and allow_overwrite isn't True.

Parameters

- **path** (*str*) – Path to a file or folder.
- **allow_overwrite** (*bool*) – Whether or not to allow overwriting existing files, by default False

Raises

- `FileExistsError` – If path points to an existing file.
- `FileExistsError` – If path points to an existing folder which is not empty.

model_registration

Model registration convenience functions.

Functions

Summary

get_model	Retrieve a model from the model registry.
is_known_model	Check if a model is in the model registry.
known_model_names	Names of the registered models.
model_plugin_table	Return registered model plugins as markdown table.
register_model	Add a model to the model registry.
set_model_plugin	Set the plugin used for a specific model name.

get_model

glotaran.plugin_system.model_registration.**get_model**(*model_type: str*) → type[Model]

Retrieve a model from the model registry.

Parameters **model_type** (*str*) – Name of the model under which it is registered.

Returns Model class

Return type type[*Model*]

is_known_model

glotaran.plugin_system.model_registration.**is_known_model**(*model_type: str*) → bool

Check if a model is in the model registry.

Parameters **model_type** (*str*) – Name of the model under which it is registered.

Returns Whether or not the model is registered.

Return type bool

known_model_names

glotaran.plugin_system.model_registration.**known_model_names**(*full_names: bool = False*) → list[str]

Names of the registered models.

Parameters **full_names** (*bool*) – Whether to display the full names the plugins are registered under as well.

Returns List of registered models.

Return type list[str]

model_plugin_table

glotaran.plugin_system.model_registration.**model_plugin_table**(**, plugin_names: bool = False, full_names: bool = False*) → *glotaran.utils.ipython.MarkdownStr*

Return registered model plugins as markdown table.

This is especially useful when you work with new plugins.

Parameters

- **plugin_names** (*bool*) – Whether or not to add the names of the plugins to the table.
- **full_names** (*bool*) – Whether to display the full names the plugins are registered under as well.

Returns Markdown table of modelnames.

Return type str

register_model

glotaran.plugin_system.model_registration.**register_model**(*model_type: str, model: type[Model]*) → None

Add a model to the model registry.

Parameters

- **model_type** (*str*) – Name of the model under which it is registered.
- **model** (*type[Model]*) – model class to be registered.

set_model_plugin

glotaran.plugin_system.model_registration.**set_model_plugin**(*model_name: str, full_plugin_name: str*) → None

Set the plugin used for a specific model name.

This function is useful when you want to resolve conflicts of installed plugins or overwrite the plugin used for a specific model name.

Effected functions:

- optimize()

Parameters

- **model_name** (*str*) – Name of the model to use the plugin for.
- **full_plugin_name** (*str*) – Full name (import path) of the registered plugin.

project_io_registration

Project Io registration convenience functions.

Note: The [call-arg] type error would be raised since the base methods doesn't have a `**kwargs` argument, but we rather ignore this error here, than adding `**kwargs` to the base method and causing an [override] type error in the plugins implementation.

Functions

Summary

get_project_io	Retrieve a data io plugin from the project_io registry.
get_project_io_method	Retrieve implementation of project io functionality for the format 'format_name'.
is_known_project_format	Check if a data format is in the project_io registry.
known_project_formats	Names of the registered project io plugins.
load_model	Create a Model instance from the specs defined in a file.
load_parameters	Create a `ParameterGroup` instance from the specs defined in a file.
load_result	Create a `Result` instance from the specs defined in a file.
load_scheme	Create a `Scheme` instance from the specs defined in a file.
project_io_plugin_table	Return registered project io plugins and which functions they support as markdown table.
register_project_io	Register project io plugins to one or more formats.
save_model	Save a `Model` instance to a spec file.
save_parameters	Save a `ParameterGroup` instance to a spec file.
save_result	Write a `Result` instance to a spec file.
save_scheme	Save a `Scheme` instance to a spec file.
set_project_plugin	Set the plugin used for a specific project format.
show_project_io_method_help	Show help for the implementation of project io plugin methods.

get_project_io

`glotaran.plugin_system.project_io_registration.`**`get_project_io`**(*format_name: str*) → *glotaran.io.interface.ProjectIoInterface*

Retrieve a data io plugin from the project_io registry.

Parameters **`format_name`** (*str*) – Name of the data io plugin under which it is registered.

Returns Project io plugin instance.

Return type *ProjectIoInterface*

get_project_io_method

`glotaran.plugin_system.project_io_registration.`**`get_project_io_method`**(*format_name: str*, *method_name: ProjectIoMethods*) → Callable[..., Any]

Retrieve implementation of project io functionality for the format 'format_name'.

This allows to get the proper help and autocomplete for the function, which is especially valuable if the function provides additional options.

Parameters

- **`format_name`** (*str*) – Format the dataloader should be able to read.
- **`method_name`** (*{'load_model', 'write_model', 'load_parameters', 'write_parameters', 'load_scheme', 'write_scheme', 'load_result', 'write_result'}*) – Method name, e.g. load_model.

Returns The function which is called in the background by the convenience functions.

Return type Callable[. . . , Any]

is_known_project_format

`glotaran.plugin_system.project_io_registration.`**`is_known_project_format`**(*format_name: str*) → bool

Check if a data format is in the project_io registry.

Parameters **`format_name`** (*str*) – Name of the project io plugin under which it is registered.

Returns Whether or not the data format is a registered project io plugin.

Return type bool

known_project_formats

glotaran.plugin_system.project_io_registration.**known_project_formats**(*full_names: bool = False*) → list[str]

Names of the registered project io plugins.

Parameters **full_names** (*bool*) – Whether to display the full names the plugins are registered under as well.

Returns List of registered project io plugins.

Return type list[str]

load_model

glotaran.plugin_system.project_io_registration.**load_model**(*file_name: str | PathLike[str], format_name: str = None, **kwargs: Any*) → Model

Create a Model instance from the specs defined in a file.

Parameters

- **file_name** (*str | PathLike[str]*) – File containing the model specs.
- **format_name** (*str*) – Format the file is in, if not provided it will be inferred from the file extension.
- ****kwargs** (*Any*) – Additional keyword arguments passes to the `load_model` implementation of the project io plugin.

Returns Model instance created from the file.

Return type *Model*

load_parameters

glotaran.plugin_system.project_io_registration.**load_parameters**(*file_name: str | PathLike[str], format_name: str = None, **kwargs*) → ParameterGroup

Create a `ParameterGroup` instance from the specs defined in a file.

Parameters

- **file_name** (*str | PathLike[str]*) – File containing the parameter specs.
- **format_name** (*str*) – Format the file is in, if not provided it will be inferred from the file extension.
- ****kwargs** (*Any*) – Additional keyword arguments passes to the `load_parameters` implementation of the project io plugin.

Returns `ParameterGroup` instance created from the file.

Return type *ParameterGroup*

load_result

`glotaran.plugin_system.project_io_registration.`**`load_result`**(*result_path: str | PathLike[str], format_name: str = None, **kwargs: Any*) → Result

Create a `Result` instance from the specs defined in a file.

Parameters

- **`result_path`** (*str | PathLike[str]*) – Path containing the result data.
- **`format_name`** (*str*) – Format the result is in, if not provided and it is a file it will be inferred from the file extension.
- **`**kwargs`** (*Any*) – Additional keyword arguments passes to the `load_result` implementation of the project io plugin.

Returns `Result` instance created from the saved format.

Return type *Result*

load_scheme

`glotaran.plugin_system.project_io_registration.`**`load_scheme`**(*file_name: str | PathLike[str], format_name: str = None, **kwargs: Any*) → Scheme

Create a `Scheme` instance from the specs defined in a file.

Parameters

- **`file_name`** (*str | PathLike[str]*) – File containing the parameter specs.
- **`format_name`** (*str*) – Format the file is in, if not provided it will be inferred from the file extension.
- **`**kwargs`** (*Any*) – Additional keyword arguments passes to the `load_scheme` implementation of the project io plugin.

Returns `Scheme` instance created from the file.

Return type *Scheme*

project_io_plugin_table

`glotaran.plugin_system.project_io_registration.`**`project_io_plugin_table`**(**, plugin_names: bool = False, full_names: bool = False*) → *glotaran.utils.ipython.MarkdownStr*

Return registered project io plugins and which functions they support as markdown table.

This is especially useful when you work with new plugins.

Parameters

- **plugin_names** (*bool*) – Whether or not to add the names of the plugins to the table.
- **full_names** (*bool*) – Whether to display the full names the plugins are registered under as well.

Returns Markdown table of project io plugins.

Return type str

register_project_io

glotaran.plugin_system.project_io_registration.**register_project_io**(*format_names: str | list[str]*) → Callable[[type[ProjectIoInterface]], type[ProjectIoInterface]]

Register project io plugins to one or more formats.

Decorate a project io plugin class with `@register_project_io(format_name|[*format_names])` to add it to the registry.

Parameters **format_names** (*str | list[str]*) – Name of the project io plugin under which it is registered.

Returns Inner decorator function.

Return type Callable[[type[*ProjectIoInterface*]], type[*ProjectIoInterface*]]

Examples

```
>>> @register_project_io("my_format_1")
... class MyProjectIo1(ProjectIoInterface):
...     pass
```

```
>>> @register_project_io(["my_format_1", "my_format_1_alias"])
... class MyProjectIo2(ProjectIoInterface):
...     pass
```

save_model

glotaran.plugin_system.project_io_registration.**save_model**(*model: Model, file_name: str | PathLike[str], format_name: str = None, *, allow_overwrite: bool = False, **kwargs: Any*) → None

Save a `Model` instance to a spec file.

Parameters

- **model** (`Model`) – `Model` instance to save to specs file.
- **file_name** (*str | PathLike[str]*) – File to write the model specs to.

- **format_name** (*str*) – Format the file should be in, if not provided it will be inferred from the file extension.
- **allow_overwrite** (*bool*) – Whether or not to allow overwriting existing files, by default False
- ****kwargs** (*Any*) – Additional keyword arguments passes to the `save_model` implementation of the project io plugin.

save_parameters

`glotaran.plugin_system.project_io_registration.`**`save_parameters`**(*parameters: ParameterGroup*, *file_name: str | PathLike[str]*, *format_name: str = None*, ***, *allow_overwrite: bool = False*, ***kwargs: Any*) → None

Save a `ParameterGroup` instance to a spec file.

Parameters

- **parameters** (`ParameterGroup`) – `ParameterGroup` instance to save to specs file.
- **file_name** (*str | PathLike[str]*) – File to write the parameter specs to.
- **format_name** (*str*) – Format the file should be in, if not provided it will be inferred from the file extension.
- **allow_overwrite** (*bool*) – Whether or not to allow overwriting existing files, by default False
- ****kwargs** (*Any*) – Additional keyword arguments passes to the `save_parameters` implementation of the project io plugin.

save_result

`glotaran.plugin_system.project_io_registration.`**`save_result`**(*result: Result*, *result_path: str | PathLike[str]*, *format_name: str = None*, ***, *allow_overwrite: bool = False*, ***kwargs: Any*) → None

Write a `Result` instance to a spec file.

Parameters

- **result** (`Result`) – `Result` instance to write.
- **result_path** (*str | PathLike[str]*) – Path to write the result data to.
- **format_name** (*str*) – Format the result should be saved in, if not provided and it is a file it will be inferred from the file extension.

- **allow_overwrite** (*bool*) – Whether or not to allow overwriting existing files, by default False
- ****kwargs** (*Any*) – Additional keyword arguments passes to the `save_result` implementation of the project io plugin.

save_scheme

`glotaran.plugin_system.project_io_registration.`**`save_scheme`**(*scheme: Scheme, file_name: str | PathLike[str], format_name: str = None, *, allow_overwrite: bool = False, **kwargs: Any*) → None

Save a `Scheme` instance to a spec file.

Parameters

- **scheme** (*Scheme*) – `Scheme` instance to save to specs file.
- **file_name** (*str | PathLike[str]*) – File to write the scheme specs to.
- **format_name** (*str*) – Format the file should be in, if not provided it will be inferred from the file extension.
- **allow_overwrite** (*bool*) – Whether or not to allow overwriting existing files, by default False
- ****kwargs** (*Any*) – Additional keyword arguments passes to the `save_scheme` implementation of the project io plugin.

set_project_plugin

`glotaran.plugin_system.project_io_registration.`**`set_project_plugin`**(*format_name: str, full_plugin_name: str*) → None

Set the plugin used for a specific project format.

This function is useful when you want to resolve conflicts of installed plugins or overwrite the plugin used for a specific format.

Effected functions:

- *load_model()*
- *save_model()*
- *load_parameters()*
- *save_parameters()*
- *load_scheme()*
- *save_scheme()*
- *load_result()*
- *save_result()*

Parameters

- **format_name** (*str*) – Format name used to refer to the plugin when used for `save` and `load` functions.
- **full_plugin_name** (*str*) – Full name (import path) of the registered plugin.

show_project_io_method_help

glotaran.plugin_system.project_io_registration.**show_project_io_method_help**(*format_name: str*, *method_name: ProjectIoMethods*) → None

Show help for the implementation of project io plugin methods.

Parameters

- **format_name** (*str*) – Format the method should support.
- **method_name** (*{'load_model', 'write_model', 'load_parameters', 'write_parameters', 'load_scheme', 'write_scheme', 'load_result', 'write_result'}*) – Method name.

12.1.10 project

Modules

glotaran.project.result	The result class for global analysis.
glotaran.project.scheme	

result

The result class for global analysis.

Classes

Summary

Result	The result of a global analysis

Result

class glotaran.project.result.**Result**(*additional_penalty: np.ndarray | None, cost: ArrayLike, data: dict[str, xr.Dataset], free_parameter_labels: list[str], number_of_function_evaluations: int, initial_parameters: ParameterGroup, optimized_parameters: ParameterGroup, scheme: Scheme, success: bool, termination_reason: str, chi_square: float | None = None, covariance_matrix: ArrayLike | None = None, degrees_of_freedom: int | None = None, jacobian: ArrayLike | None = None, number_of_data_points: int | None = None, number_of_jacobian_evaluations: int | None = None, number_of_variables: int | None = None, optimality: float | None = None, reduced_chi_square: float | None = None, root_mean_square_error: float | None = None*)

Bases: `object`

The result of a global analysis

Attributes Summary

chi_square	The chi-square of the optimization.
covariance_matrix	Covariance matrix.
degrees_of_freedom	Degrees of freedom in optimization $N - N_{vars}$.
jacobian	Modified Jacobian matrix at the solution
model	
number_of_data_points	Number of data points N.
number_of_jacobian_evaluations	The number of jacobian evaluations.
number_of_variables	Number of variables in optimization N_{vars}
optimality	
reduced_chi_square	The reduced chi-square of the optimization.
root_mean_square_error	The root mean square error the optimization.

chi_square

Result.**chi_square: float | None = None**

The chi-square of the optimization.

$\chi^2 = \sum_i^N [Residual_i]^2$.

covariance_matrix

`Result.covariance_matrix: ArrayLike | None = None`

Covariance matrix.

The rows and columns are corresponding to *free_parameter_labels*.

degrees_of_freedom

`Result.degrees_of_freedom: int | None = None`

Degrees of freedom in optimization $N - N_{vars}$.

jacobian

`Result.jacobian: ArrayLike | None = None`

Modified Jacobian matrix at the solution

See also: `scipy.optimize.least_squares()`

model

`Result.model`

number_of_data_points

`Result.number_of_data_points: int | None = None`

Number of data points N.

number_of_jacobian_evaluations

`Result.number_of_jacobian_evaluations: int | None = None`

The number of jacobian evaluations.

number_of_variables

`Result.number_of_variables: int | None = None`

Number of variables in optimization N_{vars}

optimality

`Result.optimality: float | None = None`

reduced_chi_square

`Result.reduced_chi_square: float | None = None`

The reduced chi-square of the optimization.

$\chi^2_{red} = \chi^2/(N - N_{vars})$.

root_mean_square_error

`Result.root_mean_square_error: float | None = None`

The root mean square error the optimization.

$rms = \sqrt{\chi^2_{red}}$

Methods Summary

get_dataset	Returns the result dataset for the given dataset label.
get_scheme	Return a new scheme from the Result object with optimized parameters.
markdown	Formats the model as a markdown text.
save	Saves the result to given folder.

get_dataset

`Result.get_dataset`(*dataset_label: str*) → xarray.core.dataset.Dataset

Returns the result dataset for the given dataset label.

> **Warning:** Deprecated use `glotaran.project.result.Result.data[dataset_label]` instead.

Parameters **dataset_label** – The label of the dataset.

get_scheme

`Result.get_scheme()` → *glotaran.project.scheme.Scheme*

Return a new scheme from the Result object with optimized parameters.

Returns A new scheme with the parameters set to the optimized values. For the dataset weights the (precomputed) weights from the original scheme are used.

Return type *Scheme*

markdown

`Result.markdown`(*with_model: bool = True*, *base_heading_level: int = 1*) → *glotaran.utils.ipython.MarkdownStr*

Formats the model as a markdown text.

Parameters `with_model` – If *True*, the model will be printed with initial and optimized parameters filled in.

save

`Result.save`(*path: str*) → list[str]

Saves the result to given folder.

> **Warning:** Deprecated use `save_result(result_path=result_path, result=result, format_name="legacy", allow_overwrite=True)` instead.

Returns a list with paths of all saved items. The following files are saved:

- *result.md*: The result with the model formatted as markdown text.
- *optimized_parameters.csv*: The optimized parameter as csv file.
- *{dataset_label}.nc*: The result data for each dataset as NetCDF file.

Parameters `path` – The path to the folder in which to save the result.

Methods Documentation

`additional_penalty: np.ndarray | None`

A vector with the value for each additional penalty, or None

`chi_square: float | None = None`

The chi-square of the optimization.

$\chi^2 = \sum_i^N [Residual_i]^2$.

`cost: ArrayLike`

`covariance_matrix: ArrayLike | None = None`

Covariance matrix.

The rows and columns are corresponding to *`free_parameter_labels`*.

`data: dict[str, xr.Dataset]`

The resulting data as a dictionary of xarray.Dataset.

Notes

The actual content of the data depends on the actual model and can be found in the documentation for the model.

degrees_of_freedom: int | None = None
Degrees of freedom in optimization $N - N_{vars}$.

free_parameter_labels: list[str]
List of labels of the free parameters used in optimization.

get_dataset(*dataset_label: str*) → xarray.core.dataset.Dataset
Returns the result dataset for the given dataset label.

> **Warning:** Deprecated use `glotaran.project.result.Result.data[dataset_label]` instead.

Parameters **dataset_label** – The label of the dataset.

get_scheme() → *glotaran.project.scheme.Scheme*
Return a new scheme from the Result object with optimized parameters.

Returns A new scheme with the parameters set to the optimized values. For the dataset weights the (precomputed) weights from the original scheme are used.

Return type *Scheme*

initial_parameters: ParameterGroup

jacobian: ArrayLike | None = None
Modified Jacobian matrix at the solution

See also: `scipy.optimize.least_squares()`

markdown(*with_model: bool = True*, *base_heading_level: int = 1*) → *glotaran.utils.ipython.MarkdownStr*
Formats the model as a markdown text.

Parameters **with_model** – If *True*, the model will be printed with initial and optimized parameters filled in.

property model: glotaran.model.base_model.Model

number_of_data_points: int | None = None
Number of data points N.

number_of_function_evaluations: int
The number of function evaluations.

number_of_jacobian_evaluations: int | None = None
The number of jacobian evaluations.

number_of_variables: int | None = None
Number of variables in optimization N_{vars}

optimality: float | None = None

optimized_parameters: ParameterGroup
The optimized parameters, organized in a `ParameterGroup`

reduced_chi_square: float | None = None
The reduced chi-square of the optimization.

$\chi^2_{red} = \chi^2/(N - N_{vars})$.

root_mean_square_error: float | None = None
The root mean square error the optimization.

$rms = \sqrt{\chi^2_{red}}$

save(*path: str*) → list[str]
Saves the result to given folder.

> **Warning:** Deprecated use `save_result(result_path=result_path, result=result, format_name="legacy", allow_overwrite=True)` instead.

Returns a list with paths of all saved items. The following files are saved:

- *result.md*: The result with the model formatted as markdown text.
- *optimized_parameters.csv*: The optimized parameter as csv file.
- *{dataset_label}.nc*: The result data for each dataset as NetCDF file.

Parameters **path** – The path to the folder in which to save the result.

scheme: Scheme

success: bool
Indicates if the optimization was successful.

termination_reason: str
The reason (message when) the optimizer terminated

scheme

Classes

Summary

SavingOptions
Scheme

SavingOptions

class glotaran.project.scheme.**SavingOptions**(*level: Literal[('minimal', 'full')] = 'full', data_filter: list[str] | None = None, data_format: str = 'nc', parameter_format: str = 'csv', report: bool = True*)

Bases: object

Attributes Summary

data_filter
data_format
level
parameter_format
report

data_filter

SavingOptions.**data_filter: list[str] | None = None**

data_format

SavingOptions.**data_format: str = 'nc'**

level

SavingOptions.**level: Literal[minimal, full] = 'full'**

parameter_format

SavingOptions.**parameter_format: str = 'csv'**

report

SavingOptions.**report: bool = True**

Methods Summary

—

Methods Documentation

data_filter: list[str] | None = None

data_format: str = 'nc'

level: Literal[minimal, full] = 'full'

parameter_format: str = 'csv'

report: bool = True

Scheme

class glotaran.project.scheme.**Scheme**(*model: Model | str, parameters: ParameterGroup | str, data: dict[str, xr.DataArray | xr.Dataset | str], group_tolerance: float = 0.0, non_negative_least_squares: bool = False, maximum_number_function_evaluations: int = None, ftol: float = 1e-08, gtol: float = 1e-08, xtol: float = 1e-08, optimization_method: Literal[('TrustRegionReflection', 'Dogbox', 'Levenberg-Marquardt')] = 'TrustRegionReflection', saving: SavingOptions = SavingOptions(level='full', data_filter=None, data_format='nc', parameter_format='csv', report=True), result_path: str | None = None*)

Bases: object

Attributes Summary

ftol
group_tolerance
gtol
maximum_number_function_evaluations
non_negative_least_squares
optimization_method
result_path
saving

continues on next page

Table 209 – continued from previous page

xtol

ftol

`Scheme.ftol: float = 1e-08`

group_tolerance

`Scheme.group_tolerance: float = 0.0`

gtol

`Scheme.gtol: float = 1e-08`

maximum_number_function_evaluations

`Scheme.maximum_number_function_evaluations: int = None`

non_negative_least_squares

`Scheme.non_negative_least_squares: bool = False`

optimization_method

`Scheme.optimization_method: Literal[TrustRegionReflection, Dogbox, Levenberg-Marquardt] = 'TrustRegionReflection'`

result_path

`Scheme.result_path: str | None = None`

saving

`Scheme.saving: SavingOptions = SavingOptions(level='full', data_filter=None, data_format='nc', parameter_format='csv', report=True)`

xtol

`Scheme.xtol: float = 1e-08`

Methods Summary

from_yaml_file	Create *Scheme* from specs in yaml file.
markdown	Formats the *Scheme* as markdown string.
problem_list	Returns a list with all problems in the model and missing parameters.
valid	Returns *True* if there are no problems with the model or the parameters, else *False*.
validate	Returns a string listing all problems in the model and missing parameters.

from_yaml_file

static `Scheme.from_yaml_file`(*filename: str*) → *glotaran.project.scheme.Scheme*

Create *Scheme* from specs in yaml file.

> **Warning:** Deprecated use `glotaran.io.load_scheme(filename)` instead.

Parameters **filename** (*str*) – Path to the spec file.

Returns Analysis schmeme

Return type *Scheme*

markdown

`Scheme.markdown()`

Formats the *Scheme* as markdown string.

problem_list

`Scheme.problem_list()` → list[str]

Returns a list with all problems in the model and missing parameters.

valid

`Scheme.valid(parameters: ParameterGroup = None) → bool`
Returns *True* if there are no problems with the model or the parameters, else *False*.

validate

`Scheme.validate() → str`
Returns a string listing all problems in the model and missing parameters.

Methods Documentation

data: dict[str, xr.DataArray | xr.Dataset | str]

static from_yaml_file(*filename: str*) → *glotaran.project.scheme.Scheme*
Create `Scheme` from specs in yaml file.

> **Warning:** Deprecated use `glotaran.io.load_scheme(filename)` instead.

Parameters **filename** (*str*) – Path to the spec file.

Returns Analysis schmeme

Return type *Scheme*

ftol: float = 1e-08

group_tolerance: float = 0.0

gtol: float = 1e-08

markdown()
Formats the `Scheme` as markdown string.

maximum_number_function_evaluations: int = None

model: Model | str

non_negative_least_squares: bool = False

optimization_method: Literal[TrustRegionReflection, Dogbox, Levenberg-Marquardt] = 'TrustRegionReflection'

parameters: ParameterGroup | str

problem_list() → list[str]
Returns a list with all problems in the model and missing parameters.

result_path: str | None = None

saving: *SavingOptions* = SavingOptions(level='full', data_filter=None, data_format='nc', parameter_format='csv', report=True)

valid(*parameters: ParameterGroup = None*) → bool
Returns *True* if there are no problems with the model or the parameters, else *False*.

validate() → str
Returns a string listing all problems in the model and missing parameters.

`xtol: float = 1e-08`

12.1.11 utils

Glotaran utility function/class package.

Modules

glotaran.utils.ipython	Glotaran module with utilities for `ipython` integration (e.g.

ipython

Glotaran module with utilities for `ipython` integration (e.g. `notebooks`).

Functions

Summary

display_file	Display a file with syntax highlighting `syntax`.

display_file

`glotaran.utils.ipython.`**`display_file`**(*path: str | PathLike[str]*, ***, *syntax: str = None*) → MarkdownStr

Display a file with syntax highlighting `syntax`.

Parameters

- **path** (*str | PathLike[str]*) – Paths to the file
- **syntax** (*str, optional*) – Syntax highlighting which should be applied, by default None

Returns File content with syntax highlighting to render in ipython.

Return type *MarkdownStr*

Classes

Summary

MarkdownStr	String wrapper class for rich display integration of markdown in ipython.

MarkdownStr

class glotaran.utils.ipython.**MarkdownStr**(*wrapped_str: str*, **, *syntax: Optional[str] = None*)

Bases: `collections.UserString`

String wrapper class for rich display integration of markdown in ipython.

String class automatically displayed as markdown by ipython.

Parameters

- **wrapped_str** (*str*) – String to be wrapped.
- **syntax** (*str*) – Syntax highlighting which should be applied, by default None

Note: Possible syntax highlighting values can e.g. be found here: https://support.codebasehq.com/articles/tips-tricks/syntax-highlighting-in-markdown

Methods Summary

capitalize	
casefold	
center	
count	
encode	
endswith	
expandtabs	
find	
format	
format_map	
index	Raises ValueError if the value is not present.
isalnum	
isalpha	
isascii	
isdecimal	
isdigit	

continues on next page

Table 214 – continued from previous page

isidentifier	
islower	
isnumeric	
isprintable	
isspace	
istitle	
isupper	
join	
ljust	
lower	
lstrip	
maketrans	Return a translation table usable for str.translate().
partition	
replace	
rfind	
rindex	
rjust	
rpartition	
rsplit	
rstrip	
split	
splitlines	
startswith	
strip	
swapcase	

continues on next page

Table 214 – continued from previous page

title
translate
upper
zfill

capitalize

MarkdownStr.**capitalize**()

casefold

MarkdownStr.**casefold**()

center

MarkdownStr.**center**(*width*, **args*)

count

MarkdownStr.**count**(*value*) → integer – return number of occurrences of value

encode

MarkdownStr.**encode**(*encoding='utf-8'*, *errors='strict'*)

endswith

MarkdownStr.**endswith**(*suffix*, *start=0*, *end=9223372036854775807*)

expandtabs

MarkdownStr.**expandtabs**(*tabsize=8*)

find

MarkdownStr.**find**(*sub*, *start=0*, *end=9223372036854775807*)

format

MarkdownStr.**format**(**args*, ***kwds*)

format_map

MarkdownStr.**format_map**(*mapping*)

index

MarkdownStr.**index**(*value*[, *start*[, *stop*]]) → integer – return first index of value.

Raises ValueError if the value is not present.

Supporting start and stop arguments is optional, but recommended.

isalnum

MarkdownStr.**isalnum**()

isalpha

MarkdownStr.**isalpha**()

isascii

MarkdownStr.**isascii**()

isdecimal

MarkdownStr.**isdecimal**()

isdigit

MarkdownStr.**isdigit**()

isidentifier

MarkdownStr.**isidentifier**()

islower

MarkdownStr.**islower**()

isnumeric

MarkdownStr.**isnumeric**()

isprintable

MarkdownStr.**isprintable**()

isspace

MarkdownStr.**isspace**()

istitle

MarkdownStr.**istitle**()

isupper

MarkdownStr.**isupper**()

join

MarkdownStr.**join**(*seq*)

ljust

MarkdownStr.**ljust**(*width*, **args*)

lower

MarkdownStr.**lower**()

lstrip

MarkdownStr.**lstrip**(*chars=None*)

maketrans

MarkdownStr.**maketrans**(*x*, *y=<unrepresentable>*, *z=<unrepresentable>*, */*)

Return a translation table usable for str.translate().

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in x will be mapped to the character at the same position in y. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

partition

MarkdownStr.**partition**(*sep*)

replace

`MarkdownStr.replace(old, new, maxsplit=- 1)`

rfind

`MarkdownStr.rfind(sub, start=0, end=9223372036854775807)`

rindex

`MarkdownStr.rindex(sub, start=0, end=9223372036854775807)`

rjust

`MarkdownStr.rjust(width, *args)`

rpartition

`MarkdownStr.rpartition(sep)`

rsplit

`MarkdownStr.rsplit(sep=None, maxsplit=- 1)`

rstrip

`MarkdownStr.rstrip(chars=None)`

split

`MarkdownStr.split(sep=None, maxsplit=- 1)`

splitlines

MarkdownStr.**splitlines**(*keepends=False*)

startswith

MarkdownStr.**startswith**(*prefix*, *start=0*, *end=9223372036854775807*)

strip

MarkdownStr.**strip**(*chars=None*)

swapcase

MarkdownStr.**swapcase**()

title

MarkdownStr.**title**()

translate

MarkdownStr.**translate**(**args*)

upper

MarkdownStr.**upper**()

zfill

MarkdownStr.**zfill**(*width*)

Methods Documentation

capitalize()

casefold()

center(*width*, **args*)

count(*value*) → integer – return number of occurrences of value

encode(*encoding='utf-8'*, *errors='strict'*)

endswith(*suffix*, *start=0*, *end=9223372036854775807*)

expandtabs(*tabsize=8*)

find(*sub*, *start=0*, *end=9223372036854775807*)

format(**args*, ***kwds*)

format_map(*mapping*)

index(*value*[, *start*[, *stop*]]) → integer – return first index of value.

Raises ValueError if the value is not present.

Supporting start and stop arguments is optional, but recommended.

isalnum()

isalpha()

isascii()

isdecimal()

isdigit()

isidentifier()

islower()

isnumeric()

isprintable()

isspace()

istitle()

isupper()

join(*seq*)

ljust(*width*, **args*)

lower()

lstrip(*chars=None*)

maketrans(*x*, *y=<unrepresentable>*, *z=<unrepresentable>*, */*)

Return a translation table usable for str.translate().

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in x will be mapped to the character at the same position in y. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

partition(*sep*)

replace(*old*, *new*, *maxsplit=- 1*)

rfind(*sub*, *start=0*, *end=9223372036854775807*)

rindex(*sub*, *start=0*, *end=9223372036854775807*)

rjust(*width*, **args*)

rpartition(*sep*)

rsplit(*sep=None*, *maxsplit=- 1*)

rstrip(*chars=None*)

split(*sep=None*, *maxsplit=- 1*)

splitlines(*keepends=False*)

startswith(*prefix*, *start=0*, *end=9223372036854775807*)

strip(*chars=None*)

swapcase()

title()

translate(**args*)

upper()

zfill(*width*)

CHAPTER

THIRTEEN

PLUGINS

To be as flexible as possible `pyglotaran` uses a plugin system to handle new `Models`, `DataIo` and `ProjectIo`. Those plugins can be defined by `pyglotaran` itself, the user or a 3rd party plugin package.

13.1 Builtin plugins

13.1.1 Models

- `KineticSpectrumModel`
- `KineticImageModel`

13.1.2 Data Io

Plugins reading and writing data to and from xarray.Dataset or xarray.DataArray.

- `AsciiDataIo`
- `NetCDFDataIo`
- `SdtDataIo`

13.1.3 Project Io

Plugins reading and writing, `Model`,:class:*Schema*,:class:*ParameterGroup* or `Result`.

- `YmlProjectIo`
- `CsvProjectIo`
- `FolderProjectIo`

13.2 Reproducibility and plugins

With a plugin ecosystem there always is the possibility that multiple plugins try register under the same format/name. This is why plugins are registered at least twice. Once under the name the developer intended and secondly under their full name (full import path). This allows to ensure that a specific plugin is used by manually specifying the plugin, so if someone wants to run your analysis the results will be reproducible even if they have conflicting plugins installed. You can gain all information about the installed plugins by calling the corresponding `*_plugin_table` function with both options (`plugin_names` and `full_names`) set to true. To pin a used plugin use the corresponding `set_*_plugin` function with the intended name (`format_name`/`model_name`) and the full name (`full_plugin_name`) of the plugin to use.

If you wanted to ensure that the pyglotaran builtin plugin is used for `sdt` files you could add the following lines to the beginning of your analysis code.

```
from glotaran.io import set_data_plugin
set_data_plugin("sdt", "glotaran.builtin.io.sdt.sdt_file_reader.SdtDataIo_sdt")
```

13.2.1 Models

The functions for model plugins are located in `glotaran.model` and called `model_plugin_table` and `set_model_plugin`.

13.2.2 Data Io

The functions for data io plugins are located in `glotaran.io` and called `data_io_plugin_table` and `set_data_plugin`.

13.2.3 Project Io

The functions for project io plugins are located in `glotaran.io` and called `project_io_plugin_table` and `set_project_plugin`.

13.3 3rd party plugins

Plugins not part of `pyglotaran` itself.

- Not yet, why not be the first? Tell us about your plugin and we will feature it here.

CHAPTER

FOURTEEN

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

14.1 Types of Contributions

14.1.1 Report Bugs

Report bugs at https://github.com/glotaran/pyglotaran/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

14.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

14.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

14.1.4 Write Documentation

pyglotaran could always use more documentation, whether as part of the official pyglotaran docs, in docstrings, or even on the web in blog posts, articles, and such. If you are writing docstrings please use the NumPyDoc style to write them.

14.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/glotaran/pyglotaran/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

14.2 Get Started!

Ready to contribute? Here's how to set up `pyglotaran` for local development.

1. Fork the `pyglotaran` repo on GitHub.
2. Clone your fork locally:

```
$ git clone https://github.com/<your_name_here>/pyglotaran.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv pyglotaran
(pyglotaran)$ cd pyglotaran
(pyglotaran)$ python -m pip install -r requirements_dev.txt
(pyglotaran)$ pip install -e . --process-dependency-links
```

4. Install the `pre-commit` hooks, to automatically format and check your code:

```
$ pre-commit install
```

5. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

6. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ pre-commit run -a
$ py.test
```

   Or to run all at once:

```
$ tox
```

7. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

8. Submit a pull request through the GitHub website.

Note: By default pull requests will use the template located at `.github/PULL_REQUEST_TEMPLATE.md`. But we also provide custom tailored templates located inside of `.github/PULL_REQUEST_TEMPLATE`. Sadly the GitHub Web Interface doesn't provide an easy way to select them as it does for issue templates (see this comment for more details).

To use them you need to add the following query parameters to the url when creating the pull request and hit enter:

- Feature PR: `?expand=1&template=feature_PR.md`
- Bug Fix PR: `?expand=1&template=bug_fix_PR`
- Documentation PR: `?expand=1&template=docs_PR.md`

14.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a *docstring*.
3. The pull request should work for Python 3.8 and 3.9 Check your Github Actions `https://github.com/<your_name_here>/pyglotaran/actions` and make sure that the tests pass for all supported Python versions.

14.4 Docstrings

We use numpy style docstrings, which can also be autogenerated from function/method signatures by extensions for your editor.

Some extensions for popular editors are:

- autodocstring (VS-Code)
- vim-python-docstring (Vim)

Note: If your pull request improves the docstring coverage (check `pre-commit run -a interrogate`), please raise the value of the interrogate setting `fail-under` in pyproject.toml. That way the next person will improve the docstring coverage as well and everyone can enjoy a better documentation.

Warning: As soon as all our docstrings are in proper shape we will enforce that it stays that way. If you want to check if your docstrings are fine you can use pydocstyle and darglint.

14.5 Tips

To run a subset of tests:

```
$ py.test tests.test_pyglotaran
```

14.6 Deprecations

Only maintainers are allowed to decide about deprecations, thus you should first open an issue and check back with them if they are ok with deprecating something.

To make deprecations as robust as possible and give users all needed information to adjust their code, we provide helper functions inside the module *glotaran.deprecation*.

The functions you most likely want to use are

- `deprecate()` for functions, methods and classes
- `warn_deprecated()` for call arguments
- `deprecate_module_attribute()` for module attributes
- `deprecate_submodule()` for modules

Those functions not only make it easier to deprecate something, but they also check that that deprecations will be removed when they are due and that at least the imports in the warning work. Thus all deprecations need to be tested.

Tests for deprecations should be placed in `glotaran/deprecation/modules/test` which also provides the test helper functions `deprecation_warning_on_call_test_helper` and `changed_import_test_warn`. Since the tests for deprecation are mainly for maintainability and not to test the functionality (those tests should be in the appropriate place) `deprecation_warning_on_call_test_helper` will by default just test that a `DeprecationWarning` was raised and ignore all raise `Exception` s. An exception to this rule is when adding back removed functionality (which shouldn't happen in the first place but might), which should be implemented in a file under `glotaran/deprecation/modules` and filenames should be like the relative import path from glotaran root, but with `_` instead of `.`.

E.g. `glotaran.analysis.scheme` would map to `analysis_scheme.py`

The only exceptions to this rule are the root `__init__.py` which is named `glotaran_root.py` and testing changed imports which should be placed in `test_changed_imports.py`.

14.6.1 Deprecating a Function, method or class

Deprecating a function, method or class is as easy as adding the `deprecate` decorator to it. Other decorators (e.g. `@staticmethod` or `@classmethod`) should be placed both `deprecate` in order to work.

Listing 1: glotaran/some_module.py

```
from glotaran.deprecation import deprecate

@deprecate(
    deprecated_qual_name_usage="glotaran.some_module.function_to_deprecate(filename)",
    new_qual_name_usage='glotaran.some_module.new_function(filename, format_name="legacy
↪")',
    to_be_removed_in_version="0.6.0",
```

(continues on next page)

(continued from previous page)

```
)
def function_to_deprecate(*args, **kwargs):
    ...
```

14.6.2 Deprecating a call argument

When deprecating a call argument you should use `warn_deprecated` and set the argument to deprecate to a default value (e.g. `"deprecated"`) to check against. Note that for this use case we need to set `check_qual_names=(False, False)` which will deactivate the import testing. This might not always be possible, e.g. if the argument is positional only, so it might make more sense to deprecate the whole callable, just discuss what to do with our trusted maintainers.

Listing 2: glotaran/some_module.py

```
from glotaran.deprecation import deprecate

def function_to_deprecate(args1, new_arg="new_default_behavior", deprecated_arg=
↪"deprecated", **kwargs):
    if deprecated_arg != "deprecated":
        warn_deprecated(
            deprecated_qual_name_usage="deprecated_arg",
            new_qual_name_usage='new_arg="legacy"',
            to_be_removed_in_version="0.6.0",
            check_qual_names=(False, False)
        )
        new_arg = "legacy"
    ...
```

14.6.3 Deprecating a module attribute

Sometimes it might be necessary to remove an attribute (function, class, or constant) from a module to prevent circular imports or just to streamline the API. In those cases you would use `deprecate_module_attribute` inside a module `__getattr__` function definition. This will import the attribute from the new location and return it when an import or use is requested.

Listing 3: glotaran/old_package/__init__.py

```
def __getattr__(attribute_name: str):
    from glotaran.deprecation import deprecate_module_attribute

    if attribute_name == "deprecated_attribute":
        return deprecate_module_attribute(
            deprecated_qual_name="glotaran.old_package.deprecated_attribute",
            new_qual_name="glotaran.new_package.new_attribute_name",
            to_be_removed_in_version="0.6.0",
        )

    raise AttributeError(f"module {__name__} has no attribute {attribute_name}")
```

14.6.4 Deprecating a submodule

For a better logical structure, it might be needed to move modules to a different location in the project. In those cases, you would use `deprecate_submodule`, which imports the module from the new location, add it to `sys.modules` and as an attribute to the parent package.

Listing 4: glotaran/old_package/__init__.py

```
from glotaran.deprecation import deprecate_submodule

module_name = deprecate_submodule(
    deprecated_module_name="glotaran.old_package.module_name",
    new_module_name="glotaran.new_package.new_module_name",
    to_be_removed_in_version="0.6.0",
)
```

14.7 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst), the version number only needs to be changed in `glotaran/__init__.py`.

Then make a new release on GitHub and give the tag a proper name, e.g. `0.3.0` since might be included in a citation.

Github Actions will then deploy to PyPI if the tests pass.

CHAPTER

FIFTEEN

PLUGIN DEVELOPMENT

If you don't find the plugin that fits your needs you can always write your own. This sections will explain you how and what you need to know.

In time we will also provide you with a cookiecutter template, to kickstart your new plugin for publishing as a package on PyPi.

The following section was generated from docs/source/notebooks/plugin_system/plugin_howto_write_a_io_plugin.ipynb ....

15.1 How to Write your own Io plugin

There are all kinds of different data formats, so it is quite likely that your experimental setup uses a format which isn't yet supported by a `glotaran` plugin and want to write your own `DataIo` plugin to support this format.

Since `json` is very common format (admittedly not for data, but in general) and python has builtin support for it we will use it as an example.

First let's have a look which `DataIo` plugins are already installed and which functions they support.

[1]:
```
from glotaran.io import data_io_plugin_table
```

[2]:
```
data_io_plugin_table()
```

[2]:

Format name	load_dataset	save_dataset
ascii	*	*
nc	*	*
sdt	*	/

Looks like there isn't a `json` plugin installed yet, but maybe someone else did already write one, so have a look at the `` `3rd party plugins `` list in the user docsumentation <https://pyglotaran.readthedocs.io/en/latest/user_documentation/using_plugins.html>`__ before you start writing your own plugin.

For the sake of the example, we will write our `json` plugin even if there already exists one by the time you read this.

First you need to import all needed libraries and functions.

- `from __future__ import annotations`: needed to write python 3.10 typing syntax (`|`), even with a lower python version
- `json`,`xarray`: Needed for reading and writing itself
- `DataIoInterface`: needed to subclass from, this way you get the proper type and especially signature checking
- `register_data_io`: registers the DataIo plugin under the given `format_names`

[3]:
```python
from __future__ import annotations

import json

import xarray as xr

from glotaran.io.interface import DataIoInterface
from glotaran.plugin_system.data_io_registration import register_data_io
```

`DataIoInterface` has two methods we could implement `load_dataset` and `save_dataset`, which are used by the identically named functions in `glotaran.io`.

We will just implement both for our example to be complete. the quickest way to get started is to just copy over the code from `DataIoInterface` which already has the right signatures and some boilerplate docstrings, for the method arguments.

If the default arguments aren't enough for your plugin and you need your methods to have additional option, you can just add those. Note the * between `file_name` and `my_extra_option`, this tell python that `my_extra_option` is an keyword only argument and `mypy <https://github.com/python/mypy>`__ won't raise an `[override]` type error for changing the signature of the method. To help others who might use your plugin and your future self, it is good practice to documents what each parameter does in the methods docstring, which will be accessed by the help function.

Finally add the `@register_data_io` with the `format_name`'s you want to register the plugin to, in our case `json` and `my_json`.

Pro tip: You don't need to implement the whole functionality inside of the method itself,

[4]:
```python
@register_data_io(["json", "my_json"])
class JsonDataIo(DataIoInterface):
    """My new shiny glotaran plugin for json data io"""

    def load_dataset(
        self, file_name: str, *, my_extra_option: str = None
    ) -> xr.Dataset | xr.DataArray:
        """Read json data to xarray.Dataset


        Parameters
        ----------
        file_name : str
            File containing the data.
        my_extra_option: str
            This argument is only for demonstration
        """
        if my_extra_option is not None:
            print(f"Using my extra option loading json: {my_extra_option}")

        with open(file_name) as json_file:
            data_dict = json.load(json_file)
        return xr.Dataset.from_dict(data_dict)

    def save_dataset(
        self, dataset: xr.Dataset | xr.DataArray, file_name: str, *, my_extra_option=None
    ):
        """Write xarray.Dataset to a json file
```

(continues on next page)

(continued from previous page)

```
        Parameters
        ----------
        dataset : xr.Dataset
            Dataset to be saved to file.
        file_name : str
            File to write the result data to.
        my_extra_option: str
            This argument is only for demonstration
        """
        if my_extra_option is not None:
            print(f"Using my extra option for writing json: {my_extra_option}")

        data_dict = dataset.to_dict()
        with open(file_name, "w") as json_file:
            json.dump(data_dict, json_file)
```

Let's verify that our new plugin was registered successfully under the `format_names` `json` and `my_json`.

[5]:
```
data_io_plugin_table()
```

[5]:

Format name	load_dataset	save_dataset
ascii	*	*
json	*	*
my_json	*	*
nc	*	*
sdt	*	/

Now let's use the example data from the quickstart to test the reading and writing capabilities of our plugin.

[6]:
```
from glotaran.examples.sequential import dataset
from glotaran.io import load_dataset
from glotaran.io import save_dataset
```

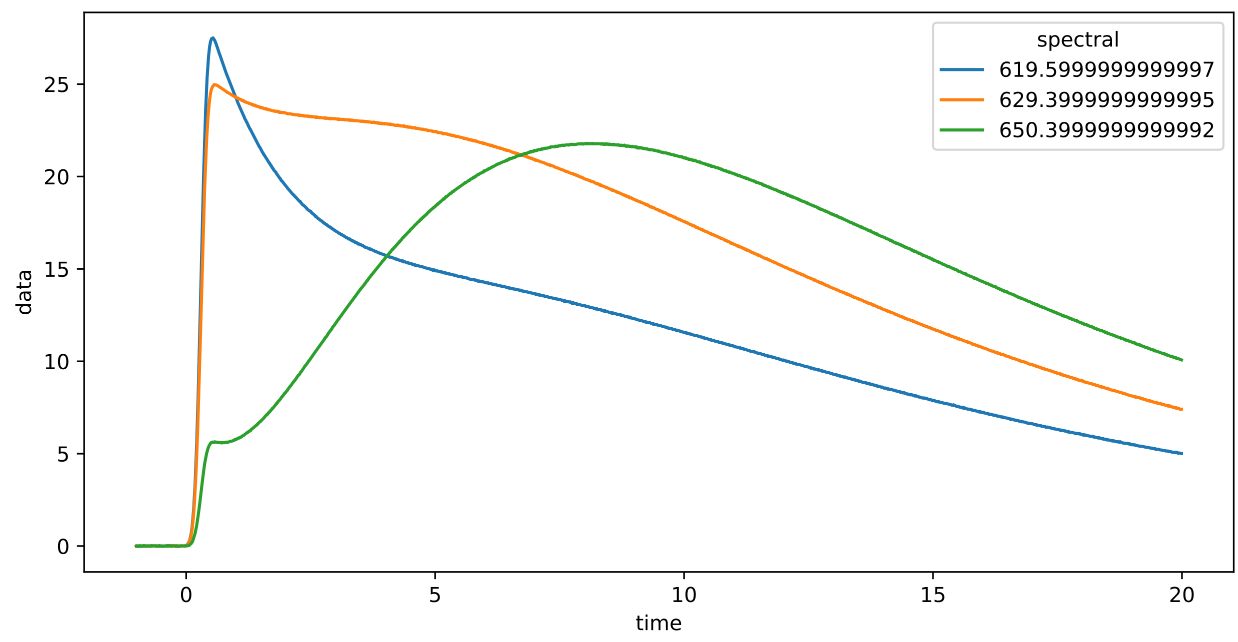
[7]:
```
dataset
```

[7]:
```
<xarray.Dataset>
Dimensions:   (spectral: 72, time: 2100)
Coordinates:
  * time      (time) float64 -1.0 -0.99 -0.98 -0.97 ... 19.96 19.97 19.98 19.99
  * spectral  (spectral) float64 600.0 601.4 602.8 604.2 ... 696.6 698.0 699.4
Data variables:
    data      (time, spectral) float64 0.003305 0.002727 ... 1.713 1.53
```

To get a feeling for our data, let's plot some traces.

[8]:
```
plot_data = dataset.data.sel(spectral=[620, 630, 650], method="nearest")
plot_data.plot.line(x="time", aspect=2, size=5)
```

[8]:
```
[<matplotlib.lines.Line2D at 0x7f7012fa36d0>,
 <matplotlib.lines.Line2D at 0x7f7012fa3700>,
 <matplotlib.lines.Line2D at 0x7f7012fa3760>]
```



Since we want to see a difference of our saved and loaded data, we divide the amplitudes by 2 for no reason.

[9]:
```
dataset["data"] = dataset.data / 2
```

Now that we changed the data, let's write them to a file.

But in which order were the arguments again? And are there any additional option?

Good thing we documented our new plugin, so we can just lookup the help.

[10]:
```
from glotaran.io import show_data_io_method_help

show_data_io_method_help("json", "save_dataset")
```

```
Help on method save_dataset in module __main__:

save_dataset(dataset: 'xr.Dataset | xr.DataArray', file_name: 'str', *, my_extra_
↪option=None) method of __main__.JsonDataIo instance
    Write xarray.Dataset to a json file

    Parameters
    ----------
    dataset : xr.Dataset
        Dataset to be saved to file.
    file_name : str
        File to write the result data to.
    my_extra_option: str
        This argument is only for demonstration
```

Note that the **function** `save_dataset` has additional arguments:

- `format_name`: overwrites the inferred plugin selection

- `allow_overwrite`: Allows to overwrite existing files **(USE WITH CAUTION!!!)**

[11]:
```
help(save_dataset)
```

```
Help on function save_dataset in module glotaran.plugin_system.data_io_registration:

save_dataset(dataset: 'xr.Dataset | xr.DataArray', file_name: 'str | PathLike[str]',
↪format_name: 'str' = None, *, allow_overwrite: 'bool' = False, **kwargs: 'Any') ->
↪'None'
    Save data from :xarraydoc:`Dataset` or :xarraydoc:`DataArray` to a file.

    Parameters
    ----------
    dataset : xr.Dataset | xr.DataArray
        Data to be written to file.
    file_name : str | PathLike[str]
        File to write the data to.
    format_name : str
        Format the file should be in, if not provided it will be inferred from the file
↪extension.
    allow_overwrite : bool
        Whether or not to allow overwriting existing files, by default False
    **kwargs : Any
        Additional keyword arguments passes to the ``write_dataset`` implementation
        of the data io plugin. If you aren't sure about those use ``get_datawriter``
        to get the implementation with the proper help and autocomplete.
```

Since this is just an example and we don't overwrite important data we will use `allow_overwrite=True`. Also it makes writing this documentation easier, not having to manually delete the test file each time you run the cell.

[12]:
```
save_dataset(
    dataset, "half_intensity.json", allow_overwrite=True, my_extra_option="just as an
↪example"
)
```

```
Using my extra option for writing json: just as an example
```

Now let's test our data loading functionality.

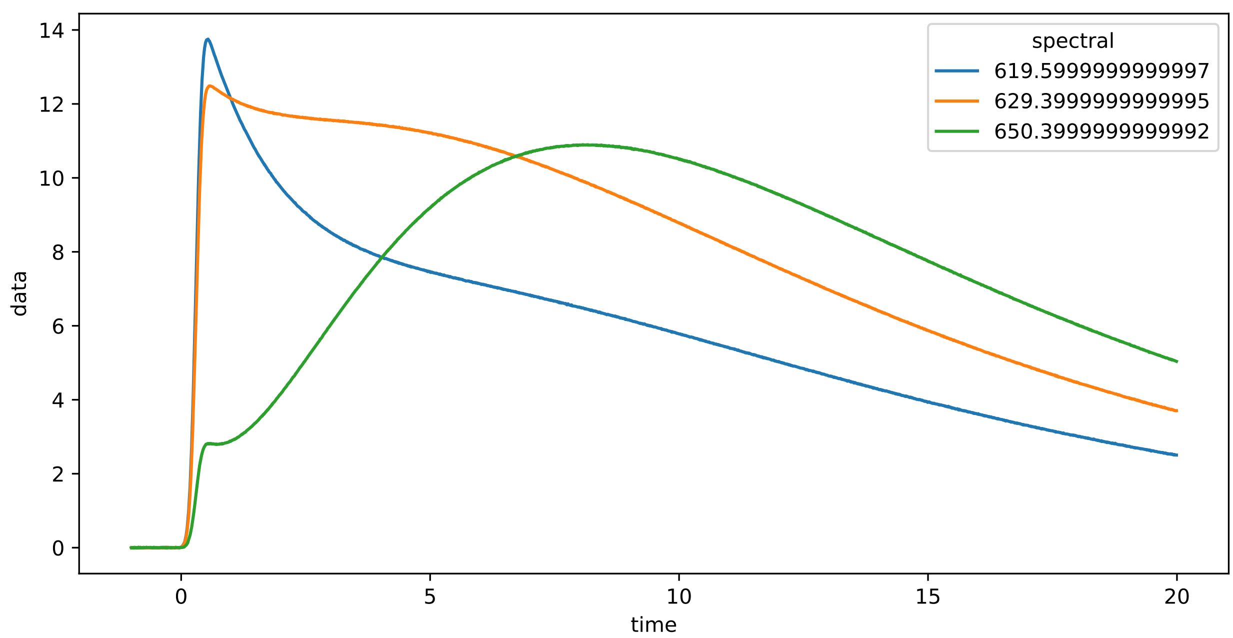
[13]:
```
reloaded_data = load_dataset("half_intensity.json", my_extra_option="just as an example")
reloaded_data
```

```
Using my extra option loading json: just as an example
```

[13]:
```
<xarray.Dataset>
Dimensions:    (spectral: 72, time: 2100)
Coordinates:
  * time       (time) float64 -1.0 -0.99 -0.98 -0.97 ... 19.96 19.97 19.98 19.99
  * spectral   (spectral) float64 600.0 601.4 602.8 604.2 ... 696.6 698.0 699.4
Data variables:
    data       (time, spectral) float64 0.001653 0.001363 ... 0.8567 0.7648
```

[14]:
```
reloaded_plot_data = reloaded_data.data.sel(spectral=[620, 630, 650], method="nearest")
reloaded_plot_data.plot.line(x="time", aspect=2, size=5)
```

[14]:
```
[<matplotlib.lines.Line2D at 0x7f70129b4400>,
 <matplotlib.lines.Line2D at 0x7f70129b4430>,
 <matplotlib.lines.Line2D at 0x7f70129b4550>]
```



Since this looks like the above plot, but with half the amplitudes, so writing and reading our data worked as we hoped it would.

Writing a `ProjectIo` plugin words analogous:

	`DataIo` plugin	`ProjectIo` plugin
Register function	`glotaran.plugin_system.data_io_registration.register_data_io`	`glotaran.plugin_system.project_io_registration.regis ter_project_io`
Baseclass	`glotaran.io.interface.DataIoInterface`	`glotaran.io.interface.Dat aIoInterface`
Possible methods	`load_dataset` , `save_dataset`	`load_model` , `save_model` , `load_parameters` , `save_parameters` , `load_scheme` , `save_scheme` , `load_result` , `save_result`

Of course you don't have to implement all methods (sometimes that doesn't even make sense), but only the ones you need.

Last but not least:

Chances are that if you need a plugin someone else does too, so it would awesome if you would publish it open source, so the wheel isn't reinvented over and over again.

CHAPTER SIXTEEN

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[1] https://glotaran.github.io/legacy/file_formats
[1] https://glotaran.github.io/legacy/file_formats

PYTHON MODULE INDEX

g

INDEX

A

D

E

F

G

H

I

J

K

L

M

N

O

S

T

U

V

W

X

Y

Z